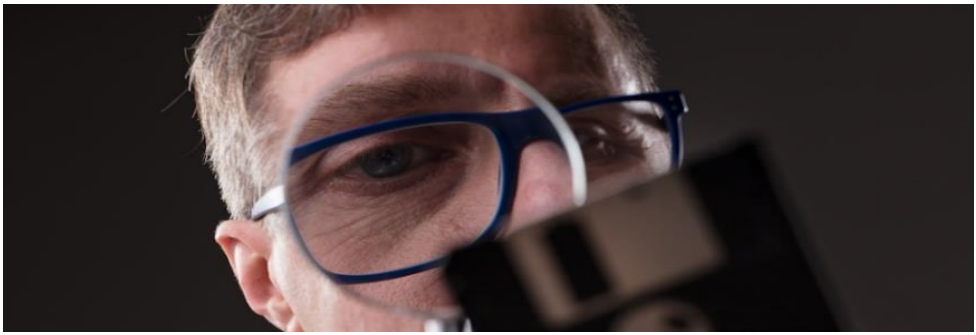
Software Maintenance: Legacy and Archaeology Event Report



This online seminar held on 6th May 2021 considered how to carry out software maintenance for legacy software that is still fulfilling a safety function. Much legacy software is not properly documented and historically poorly maintained, so what strategies are available to ensure changes can be properly assured?

Mike Parsons introduced the seminar speakers and opened by noting that many safety systems are old, yet the software still needs to be modified in a safe manner. Many changes such as security related changes can be tricky; and it is not just code modifications that are an issue; tests, documentation and associated analyses are also of concern.

It was never designed to do that!



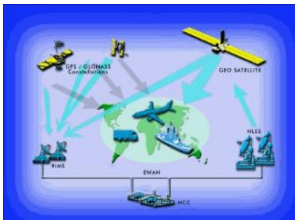
Simon Scutt from Thales, provided a brief history of the Watchkeeper Unmanned Aerial System (UAS), which was based on a design originally introduced in 1998 and commissioned almost 10 years later, with full operational capability in 2018. He noted the platform has been used successfully, but there have also been some notable accidents. Simon covered the regulatory framework and the programme organisational structure, noting that the regulator (The Military Aviation Authority) was established after the programme was started.

Simon explained that changes are classed as minor or major and that classification determines the level of approval required. All changes must be submitted by Thales as the overall coordinating design organisation with Defence Equipment & Support (DE&S) acting as the overall Technical Approval Authority. Changes are impact assessed using a proprietary change classification focussing on impact against the type certification basis.

This results in a Modification Assurance Report – something akin to a delta safety case. The level of supplier assurance is determined by the impact of the subsystem/component on airworthiness and safety, the criticality of the component and the maturity/experience of the supplier.

Independent audits are also undertaken by the regulator and the overall authority as required by DO-178B (Software Considerations in Airborne Systems and Equipment Certification) for a DAL B software system. Simon noted that safety cases tend to be deltas rather than full re-releases and concluded by saying that the main issues are with getting access to suppliers' proprietary information.

Still improving after 20 years of maintenance



Peter Niemann from CGI discussed the European Geostationary Navigation Overlay Service (EGNOS) Check Set. EGNOS is a satellite-based augmentation system providing corrections and integrity information for GPS systems and is actively being used to land aircraft. Peter described the Check Set component that checks the main processing and ultimately protects users from hazardously misleading information. The Check Set is therefore developed to documents based on Level DAL B of the DO-178B guidance.

ESA has overall responsibility for EGNOS and the service is operated by the European Satellite Service Provider (ESSP). The system was developed from 1999 with open service in 2005 and safety of life service from 2011. There has generally been one release per year and the code has grown from 60,000 to 87,000 lines over a period of 20 years. Peter discussed the main drivers for maintenance changes:

Standards Clarification: What was certifiable against DO-178B historically may not meet expectations in 2021, as a result of clarifications of ideas and objectives and global lessons learnt from its evolution to DO-178C. For example, management of derived requirements, control and data coupling, tool qualification etc. This has had a positive impact and strengthened the product with improved compiler warnings, options and exception; strengthened development processes and resulted in better tracing to address expectations.
Bugs: When defects are discovered, systematic checks for similar bugs are undertaken and analysis and lessons learnt on processes carried out.
Perfective: changes have come from algorithm enhancements and refinements.
Adaptive: hardware and software obsolescence.
Peter said that EGNOS has a strong track record of managing the challenges of changing a complex legacy system and noted the following key success factors:

- Co-operative analysis amongst all stakeholders: prototyping, analysis and evaluation before implementation eliminating unsuccessful enhancement candidates. However, the process can take 4-8 years.
- Test data selection: include live real capture where possible, regular updates based on operational data, coordinated with stakeholders, independence.
- Expertise retention: succession planning, training, up-to-date documentation, retaining expertise.
- Good practices: metrics, coding standards, reviews and configuration management.

Peter concluded that maintenance has generally improved the product but some decisions cannot be undone, for example in the use of tools and design representation.

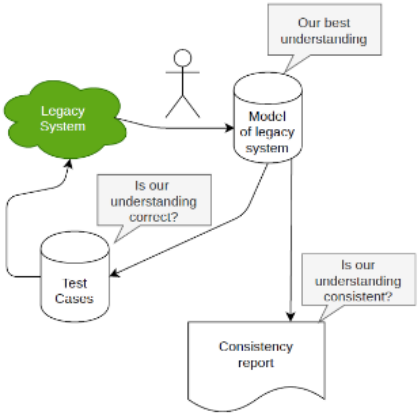
But I only changed 5 lines of code!

Chris Hobbs explained that Blackberry QNX produces operating systems and hypervisors certified to standards such as IEC 61508[1], IEC 62304[2], ISO 26262[3] and CENELEC[4] standards. Chris said the company's operating systems are widespread and can be found in many devices that we all use today, such as in cars and network routers and used in many different domains such as medical, railway and automotive. He said that although these are certified, there is a lot of legacy code, some dating back to 2002.

Chris illustrated two types of errors the "Bohrbugs"[5] – often obvious and the "Heisenbug", which only manifest themselves infrequently and under subtle timing conditions such as race conditions. He noted that even in legacy code, new bugs are still emerging, and gave a recent example where a race condition arose when the number of processors in a multicore solution was increased.

Chris referred to IEC 61508-3-1 'proven in use' requirements and said that this sets the bar high and was not usually practicable for legacy code.

He illustrated that even the simplest program has too many states to test, and gave an example of an incremental change to code that may initially yield correct results thus increasing the level of confidence, but would eventually fail in unexpected ways.



Chris asserted that testing can never be exhaustive apart from where formal proof is used, it otherwise only results in increased confidence-in-use.

Chris argued that testing can be more effective by reducing the number of states to be tested and introduced Combinatorial Testing[6], which exploits the fact that human written code has few sub-conditions in a conditional statement.

Chris concluded by describing an approach to assure significant structural changes being made to some legacy code, which was not fully documented. The approach was to build a model (and simulation) of the part of the legacy system that was to be changed. This allowed test cases to be created automatically to validate the model. A model of the changed system was then built and combined with the previous model to generate test cases and a consistency report on whether the system would meet the requirements. This revealed issues with the design, and found potential "Heisenbugs" that might otherwise have been missed.

[1] Functional safety of electrical/electronic/programmable electronic safety-related systems
[2] Medical device software — Software life cycle processes
[3] Road vehicles – Functional safety
[4] European Committee for Electrotechnical Standardization
[5] en.wikipedia.org/wiki/Heisenbug
[6] en.wikipedia.org/wiki/All-pairs_testing

Modern verification meets old code...



Rod Chapman from Protean Code Ltd has worked in a wide range of sectors, such as air traffic control, railway, military and commercial aviation and said that these tend to have software systems that are very long lived. Systems typically tend to be in service much longer than originally expected, sometimes over a decade. Rod considered two questions:

- what would he do if he was asked to reactivate and make a change to a legacy system?
- what would he do now to help future maintainers of the software in the long term?

Reactivating an old legacy system

Rod said the main problems with legacy systems are obsolescence of technology, platform, suppliers and people but modern technology can help, such as through the use of increased computer resources, including cloud, Infrastructure-as-Code (IaC), formal/static analysis and aggressive testing.

Rod said that generic compute resources are extremely cost effective now and resources can be rented not bought and easily scaled to more powerful systems as required. IaC can also setup virtual environments repeatably and make environments more readily available to developers including making representative target production environments.

Formal/Static Analysis tools have hugely improved and will find bugs but generally is "best efforts" and won't find all bugs. Application to legacy code also results in lots of warnings and false alarms. However, a "don't get any worse" strategy could be adopted, or eliminate warnings one class at a time until all are resolved.

Rod said if there is no specification or test cases then there are lots of tools now that will test code automatically and others like AddressSanitizer[7] for C/C++ that can monitor running code and detect issues such as resource usage. Some tools can simply run random data through the program (called "Fuzzing") and will find bugs.

What to do now to help future maintainers?

Rod said that the main challenge is economics – most current software has a short half-life and it is hard to make commercial decisions now that will pay off in many years and so software for the long-term is a niche market. He recommended the following:

- Beware of fashion; look for systems and companies that have a long pedigree
- Choose formal languages – their meaning doesn't change over time
- Exploit virtualisation and cloud for everything
- Freeze/snapshot virtual machines at every release point of a system
- When shutting down a project, have a re-activation plan and dry run this with someone with no previous knowledge of the system
- Commit to a long-lived architecture or deploy in a Field Programmable Gate Array
- Plan and build with multiple versions of all tools

Report by Paul Hampton, SCSC Newsletter Editor

[7] https://en.wikipedia.org/wiki/AddressSanitizer