SCS Exam M (2015-16) Solutions – Answer 3 Quesitons

1.

a) Identify the main components of risk analysis.

[5 Marks]

[Seen/Unseen problem]
We have covered these different aspects in the course but not as an enumerated list – the key issues would address: hazard identification, probability and consequence assessment, acceptability assessment, mitigation.

b) The US Department of Defence MIL-HDBK-217F on Electronic Reliability Prediction helps suppliers calculate the mean time between failure for electronic systems. It provides models for different electronic, electrical and electro-mechanical components to predict failure rates. These calculations consider environmental conditions, quality levels and stress conditions as well as the impact of non-operating periods on equipment reliability. Briefly explain why each of these factors is considered within 217F.

[5 Marks]

[Unseen problem]
Environmental conditions must be considered because extreme high or low temperatures, dust, humidity can all reduce the mean time between failures.

Quality levels are important because the design, manufacture and testing of components will have an intrinsic effect on the failure- rate.

Stress conditions will also affect mean time between failures – for example, shock and vibration can reduce the life of these components.

It is also important to consider non-operating periods because in warm of cold-standby operation then the life of the component may be extended and the mean time to failure will increase correspondingly when the item is not in use.

c) Building on your answer for part b) of this question, explain how you could go about calculation the mean time between failures for systems that include software components.
[10 Marks]

[Unseen problem]
In most cases it is not possible to calculate the mean time between failures for software. This will be influenced by a host of factors that can only be estimated – including the raw number of bugs per thousand lines of code. I have taken the class through the Musa formula that does provide one of these approximations – for instance it has an expansion ratio to account for the errors that might be introduced through the compilation from a higher elvel language to machine code etc. Some of the factors mentioned in the previous section cannot easily be accounted for in software testing – software does not age in the same way as hardware hence non-operating periods can only really influence exposure and not the burn-out time represented in the classical bath tub reliability curves.

2.

a) Briefly explain why many safety-critical applications do not use conventional approaches to multi-processing.

[5 Marks]

[Seen/Unseen problem]
Many safety-critical systems do not allow traditional forms of multi-processing. In particular, it can be hard to ensure that all safety related processes receive the computational resources they need when they are under the control of a dynamic scheduler. In some cases, a single process is run on a processor. In other cases, static scheduling using time slices will be calculated at design time to help analysts prove that safety requirements can be met. Similarly, dynamic memory allocation is usually frowned upon because one process accessing the heap may exhaust the memory used by another process – this conention may not be apparent when coding any individual process.

b) The real time operating systems used in many safety-critical applications exploit on-chip memory management units (MMU) to ensure that individual threads run in hardware-protected address spaces. The MMU will translate any attempt to access physical addresses so that they are mapped to the region associated with a thread. The MMU will also flag attempts to access illegal logical addresses. Explain the benefits that MMUs provide for safety-critical systems.
[5 Marks]

[Unseen problem]
MMUs can protect programmers from some but not all of the concerns identified in the first part of this question – the focus on memory does not address the scheduling concerns. Of course, the MMU then becomes an important single point of failure. However, it is hardware and more amenable to static analysis itself prior to running any application. The ability to monitor illegal requests is a very important aspect here because programmers can receive direct feedback so that remedial action can be taken – both at run time and also in the redesign of an offending thread.

c) The Linux operating system is gradually being introduced into a wider range of safety-critical applications, including flight control for SpaceX's Falcon, Dragon, and Grasshopper vehicles, SpaceX ground stations also run Linux. What are the main concerns about using Linux in safety-related environments?

[10 Marks]

[Unseen problem]
There are a host of concerns that must be considered but most of which can be addressed – these include:
Traceability – with open source, peer developed code can we trace the implementation of requirements through successive iterations of the code;
Training of all engineers – can we be assured of the competence of all engineers contributing to open source projects?
Repeatable processes – can we be sure that the care used in some aspects of the code will be repeated across all areas?
Proven in use – with hardware we can often argue that it has been proven in use – however, this is more difficult in software given the complexity of the code and the many millions of execution sequences;

Kernel development – we can reduce the risks by focusing on the kernel of open source code – we can look at it in detail because we have the source but this may be more and more difficult with increasing size and complexity of that kernel;
Timing issues – Linux is not by nature a real time operating system hence it is hard to express and ensure that code will meet hard deadlines;
Recovery from faults – similarly, Linux was not developed to meet complex recovery requirements following a failure;
There are many different variants or dialects of Linux hence one implementation may differ in subtle ways to another – most are aiming at meeting the POSIX target implementation requirements.

For each of these good students can discuss ways that companies such as SpaceX might address these concerns to enable the use of Linux in safety related applications.

3.

a) Briefly outline the problems that can arise when maintaining or extending the application of legacy code in safety-critical systems.

[5 Marks]

[Unseen problem]
Legacy code raises a host of concerns – lack of documentation and traceability can be compounded when the original developers no longer are with a company – subcontractors may possess the source code and associated IPR. Thee are problems when code no longer runs on modern processor architectures or when development environments and programming languages are no longer supported or when programmers who understand those languages are in short supply. Other problems arise from recreating the verification and validation conditions that were originally used to test earlier versions of the legacy code many years before etc.

b) Why can virtualization sometimes be used to extend the life of legacy code on new families of processors. What are the concerns about this approach?

[5 Marks]

[Unseen problem]
Virtualization can use software to recreate the original environment that legacy code ran on – the software emulator mimics the former processor and will itself run on more modern hardware. This raises many concerns – the software emulator may not behave exactly the same as the former environment – it can be hard to validate the correspondence between the two – especially when timing conditions are important. There may be differences in the detailed semantics. The costs of IV&V for an emulator may be greater than the costs of rewriting the legacy code etc – especially when the team developing the emulator must spend significant resources understanding both the previous and the new processor architectures. The emulator will itself have radically different failure modes than the hardware it mimics…

c) The FAA recently issued a circular on 'tool qualification guidance' that has been adapted by the European Aviation Safety Agency – this includes the following requirement:

"If your legacy system software was previously approved using ED-12 / DO-178 or ED-12A / DO-178A, and you intend to use a new or modified tool for modifications to the legacy system software,

use the criteria of ED-12C / DO-178C, section 12.2, to determine if tool qualification is needed. If you need to qualify the tool, use the software level assigned by the system safety assessment for determining the required Tool Qualification Level (TQL), and use ED-215 / DO-330 for the applicable objectives, activities, guidance, and life cycle data. You may declare your qualified tool as having satisfied ED-215 / DO-330 and not the legacy system software as having satisfied ED-12C / DO-178C".

Write a brief technical report for a project manager explaining what this means for future software development projects involving legacy code.

[10 Marks]

[Unseen problem]
There are many different solutions to this question – at its heart is the growing idea in standards such as DO-178C that you have to approve not just the code you develop but also the tools you use to develop that code – including model based development environments. This raises complex questions when legacy code might have been developed using tools that themselves did not meet the standards set out in more recent requirements. The FAA/EASA approach states that when you use new tools to modify legacy code then those tools should be assed using the more recent standards to a level appropriate to the criticality of the legacy code even though that legacy code itself might not meet all of the requirements for those standards (eg 178C).

4.

Write a technical analysis of the main challenges that arise during the safety certification and approval of software for autonomous systems.

[20 Marks]

[Essay]

As in previous questions there are many different solutions here with ample space for more able students to focus on particular examples that we have met during the course – including the lost link profile code in Predators/Reapers.

At the heart of the problem is that it can be difficult to anticipate all possible behaviors in all possible operating environments for autonomous systems. There can be emergent properties – such as flocking, when multiple autonomous systems interact or when these systems interact with groups of human controlled/conventional applications. One aspect of this is the use of dynamic, self-modifying code or systems that learn – in this situation a regulator may approve the initial version of a program but have very little idea of the eventual behaviors once an autonomous system begins a process of adaptation.

Further problems arise when regulators and certification agencies specify that an autonomous system must be "at least as safe" as a human operator. The strengths and weaknesses of these systems are very different from humans and it is typically very unclear how such a requirement could be satisfied.

In practice, there can be strong differences between the level of certainty about the operation of an autonomous system when it is being designed compared to when it has been fielded – in the pressurized context of military or police operations the use of RPAS often breaks down when controllers are no longer certain about the precise plan r set of objectives that were given to a particular mission. In theory such situations should not arise if operators followed

approved processes and procedures, however, incidents like Nogales show that reality can be very different…