Xday, XX XXX 2013.

9.30 am - 11.15am ***(check this!)***

University of Glasgow

DEGREES OF BEng, BSc, MA, MA (SOCIAL SCIENCES).

COMPUTING SCIENCE - SINGLE AND COMBINED HONOURS
ELECTRONIC AND SOFTWARE ENGINEERING - HONOURS
SOFTWARE ENGINEERING - HONOURS

SAFETY-CRITICAL SYSTEMS DEVELOPMENT

Answer 3 of the 4 questions.

1.

a) Explain why standards, such as IEC 61508, can help to improve software safety across an industry and at the same time may create "barriers to entry" that restrict competition in the development of safety-critical systems.

[5 marks]

[seen/unseen problem]

IEC 61508 is a process based standard – in other words it focuses on the techniques used to develop safety-critical software rather than focus only on testing the final product. This is justified given that testing cannot prove the absence of bugs (1 mark). Process based standards ensure that companies adopt similar approaches to the development of complex systems (1 mark). They also provide regulators with common means of assessing compliance to regulatory requirements (1 mark). However, criticisms can be made about the complexity of standards such as 61508. It can be very difficult for companies to establish the level of expertise needed to meet the requirements associated with each stage of the development lifecycle (1 mark). They may, therefore, provide barriers to new companies from entering into a market – this can be seen as a good thing when they might otherwise develop safety-critical software without adequate levels of assurance (1 mark).

b) IEC 61508 requires SIL4 systems achieve a probability of failure on demand between 1 in 10,000 and 1 in 100,000. For continuous operation the probability lies in the range 10-8 to 10-9 failures per hour of operation. Briefly explain the problems that can arise in validating the risk reductions that are associated with SIL4 systems in both continuous and on-demand mode.

Hint: there are approximately 8,760 hours in a year.

[5 marks]

[unseen problem]

As suggested in the question, IEC 61508 assumes an arbitrary one-year time interval to distinguish between low demand and high /continuous demand systems (1 mark). In each case, it is typically infeasible to exhaustively test systems to ensure that they meet these levels of reliability (1 mark). In some cases, it is possible to conduct 'fast time' simulations (1 mark) but this raises a number of questions about the veracity of the testing – will it accurately reflect the eventual context of use? (1 mark). This is a particular issue for interactive systems where operators might have to interact with real-time tests for many years or rely on an executable user model to mimic operator intervention in fast time simulations (1 mark). It is for this reason that IEC 61508 exploits the process bases approach described in the previous paragraph. Development resources are allocated in proportion to the desired level of reliability (1 mark).

c) ISO 26262 is a new automotive standard closely based on 61508; it covers the safety lifecycle from management, to development, production, operation, service and decommissioning. It addresses requirements specification, design, implementation, integration, verification, validation, and configuration using automotive risk classes known as Automotive Safety Integrity Levels (ASILs).

Write a technical report, explaining to a car producer the range of technical and organizational problems that will arise when attempting to introduce 26262 for the first time into the software that they use in their vehicles.

[10 marks]

[seen/unseen problem]

This brings together elements in the answers to the first two parts of this question – especially part a) in terms of the barriers to market entry. Key issues include the development of sufficient

competency (1 mark). Companies need to ensure they have enough staff trained to follow the processes recommended in standards such as 26262. This can be extremely expensive (1 mark). There is also a need for independent assessors who can provide feedback on progress towards the implementation of the standard (1 mark). These are not simply technical concerns – relating to significant changes in the software development process. It also creates significant challenges for management; who must ensure adequate finance and also support the external audits of development processes that ensure the implementation of the standard (1 mark).

It can be difficult to identify appropriate Automotive Safety Integrity Levels (ASILs), especially when engineers may not be familiar with the approach (1 mark). Undue conservatism may lead to increased costs that cannot be sustained. Equally, if ASILs are set at too low a level then insufficient development resources may be allocated to the development of critical code. Similar arguments can be made about the need to learn and validate the techniques associated with each of the development stages mentioned in the question (1 mark).

Further issues relate to the integration of new and existing software into the same suite of applications (1 mark). New code will have been developed under the processes that are mentioned and supported by 26262. However, most companies will have a large amount of legacy code that was not developed using this approach. The company will have to work with regulators, suppliers and customers to identify a pragmatic solution to this problem (1 mark) – for instance, treating all legacy code in the same way as COTS, with significant increased costs but without the overheads of the retrospective application of the safety standard (1 mark).

2.

a) A recent study of US Nuclear Digital Reactor Protection Systems revealed that each plant's Core Protection Calculator Systems (CPCS) used the following list of components:

- 6 computer boards
- 6 memory boards
- 4 multiplexers
- 6 watchdog timers
- 8 cold leg temperature channels
- 8 hot leg temperature channels
- 4 pressurizer pressure channels
- 4 upper core level neutron flux channels
- 4 middle core level neutron flux channels
- 4 lower core level neutron flux channels
- 4 RCP digital pump speed channels

Identify the strengths and weaknesses of both hardware and software redundancy using the CPCS as an example.

[5 marks]

[seen/unseen problem]

Hardware redundancy is an effective and commonplace technique to increase the reliability of complex, safety-critical systems (1 mark). Good solutions might sketch the bath-tub reliability curves that characterize hardware (1 mark). Redundancy provides fall-back solutions during the burn-in and burn-out phases. This is illustrated by the degree of duplication in the CPCS summarized in the question. This covers the computational hardware (processor boards) and also the sensor channels. Solutions might also mention the application of Triple Modular Redundancy (1 mark).

However, hardware redundancy relies on independence, in other words, it is effective when there are very few common cause failures that might affect multiple redundant hardware components. In contrast, software redundancy provides limited benefits because it tends not to follow the 'bath tub' probability distributions that characterize hardware (1 mark). It does not age and there are many common cause failures between multiple versions of the same code (1 mark). In other words, if one copy of a program contains a design error or bug then every other version will also contain that error. In consequence, software redundancy only really works if there is also design diversity through what is known as N-version programming (1 mark).

b) The same study examined 141 incidents involving CPCS in Digital Reactor Protection Systems. 99 involved reactor trips triggered by the CPCS for plant conditions requiring a trip or where operational errors caused the CPCS to generate a 'fail-safe' trip. In other words, the CPCS functioned correctly to protect the plant. However, 26 events involved common cause failures which delayed a trip and jeopardised safe, acceptable design limits.

Write a brief technical report identifying techniques that might be used to identify and mitigate common cause failures in complex, safety-critical software using the CPCS as an example.

[5 marks]

[unseen problem]

The previous section has mentioned N-version programming – software diversity can be used with voting to mitigate potential failures involving the CPCS. However, the question mentions common cause failures in software hence it might be assumed that these errors occurred even

when diversity was employed. In such cases, students might mention the use of a range of validation and verification techniques to improve the robustness of the code – these include walkthroughs, inspections, model based development, white and black box testing techniques, independent verification and validation (IV&V) teams, bug monitoring and reporting systems, enhanced training of programmers, the application of formal methods. Etc. (1 mark for each plausible approach up to a total of 5, an additional mark for explaining the "common cause" failure mentioned in the question and related to software).

c) A number of similarities were identified across CPCS failures. Many involved calibration errors; technicians selected the wrong data set of addressable constants and inserted them into all four CPCS channels. Others stemmed from calculation errors in generating the addressable constants – which were then loaded into the addressable constant registers. Only one event involved a software design error; processing failed sensor inputs.

Describe the relative importance of software bugs compared to calibration/configuration errors. Consider the changing importance of these sources of failure in future generations of complex control systems.

[10 marks]

[seen/unseen problem]

Safety-critical software systems are becoming increasingly complex through the integration of diverse application processes (1 mark). At the same time, significant advances have been made in the development of reliable code – through the application of model-based development, trusted compilers etc (1 mark). In consequence, it can be argued that design errors and more standard forms of bug are having less of an impact on safety-critical systems (1 mark). In many industries there are very few examples of software errors of this nature leading to loss of life (1 mark).

At the same time, configuration and calibration errors are becoming more and more significant (1 mark). The flexibility of complex software applications across a range of industries means that code can tailor control systems to optimize operational parameters in a range of different contexts (1 mark). Increased levels of automation are often only achieved when applications are set up correctly (1 mark). In consequence, removing operator interaction often simply transfers concerns over human error from direct intervention to the configuration of complex systems (1 mark).

Configuration and calibration errors can arise when operators do not adequately understand the code that they are using (1 mark) – this I particularly important when subsequent modifications are made to safety-critical systems when documentation may not adequately reflect the importance of particular parameters (1 mark). These errors can also be particularly difficult to identify when higher levels of automation mean that operators may not be continually interacting with a complex application (1 mark). For example, an incorrectly calibrated CPCS may only be identified when a trip should occur (1 mark).

3.

a) Briefly explain why it is more difficult to mitigate transient rather than intermittent or permanent failure modes involving complex software.

[5 marks]

[unseen problem]
Transient faults occur but need not recur (1 mark). They raise significant concerns because engineers cannot be sure that they will not develop into intermittent faults (1 mark), which do recur (1 mark). If there are insufficient logs or process monitoring then there may not be sufficient evidence to determine the causes or identify potential mitigations for these faults (2 marks). In contrast, permanent failures are usually much easier to diagnose and correct (1 mark).

b) Briefly explain the role of software fault injection in the verification of safety-critical systems. Your answer should distinguish between compile time injection and run-time injection. You should also distinguish between code insertion and code modification.

[5 marks]

[unseen problem]
Software fault injection deliberately introduces an error into a program to determine whether it can recover (1 mark) – for instance, through the use of watchdog timers or exception handlers (1 mark). It helps to increase confidence in the resilience of the code; however, it is critical to remove the deliberate fault prior to delivery (1 mark). Compile time injection relies on faults being introduced into the code (1 mark), run-time injection provides parameters that can be set to start the fault (1 mark). Code insertion introduces new code to create the fault (1 mark). Code modification does not introduce any new methods but relies on errors being inserted into the existing software (1 mark).

c) You have been hired by a company developing a new family of medical infusion devices. These are programmable devices that can be used to deliver fluids, medication or nutrients into a patient. Write a technical report that explains the potential limitations of software fault injection and recommends potential solutions that can be used to address the concerns that you identify for this approach to software verificaiton.

[10 marks]

[unseen problem]

The previous section has mentioned the potential danger of leaving faults in delivery code or of losing track of a previously inserted fault (1 mark). This risk can be mitigated by appropriate management practices during the final stages of development (1 mark). Further problems stem from identifying faults that can be used to test the reliability of the code in the device (1 mark). One reason for this is that software fault injection tends to be a “white box” approach; you need to understand the code that is being tested (1 mark). In consequence, developers will often introduce faults that they know their code will handle rather than failure modes that will really stress their implementation (1 mark).

Testing with independence is one mitigation; recruiting teams of test developers who have minimal links to the group implementing a system (1 mark). Alternatively, black box testing techniques might be integrated along with software fault injection (1 mark). Incident reporting systems can also be used to gather more information about the types of faults or bugs that have been seen in exiting infusion pumps (1 mark). These might then provide the basis for test case generation.

Even if independent experts cannot be directly involved in fault injection; they can help to validate the test case scenarios (1 mark). It is important not simply to focus on the types of error to be introduced into the software but also to consider the impact of those errors in a range of contexts of use – for example, with different patients in different healthcare scenarios (1 mark). This is a relatively open ended question and so a range of further alternate answers will attract full marks.

4. Human error is arguably the single greatest cause of system failure.

Write a report to senior management explaining whether it is possible to entirely eliminate human error from the design, operation and management of safety-critical software. Illustrate your answer with a range of techniques that can be used to mitigate human error and comment on the long term effectiveness of those techniques.

[20 marks]

[structured essay/unseen problem]

Previous studies have identified human error as the root cause of system failures (1 mark). Slips, lapses, mistakes are all seen as major contributors to accidents and incidents (1 mark). It is for this reason that considerable resources have been allocated to try and "engineer the operator out of complex systems" (1 mark). Driver error has led to the development of automated braking systems in rail applications. Similarly, much of the direct involvement of flight crews has been automated through the development of integrated flight management systems (1 mark).

However, this view of human error has also been challenged; for example by Hollnagel, Woods and Leveson (1 mark). They argue that humans are a key source of resilience in most safety-related systems and that operator intervention has resolved many adverse situations. If we focus only on human error then we will lose this critical perspective (1 mark). Instead, designers are urged to focus on the reasons why things went right so that we can reinforce positive behaviors rather than trying to engineer out the negative (1 mark).

Others such as Dekker have argued that undue attention has been laced on the "sharp end", on the operators who actively control complex, safety-critical systems (1 mark). Instead, more attention should be paid to the designers, managers and regulators who place operators in an environment or working context where they are more likely to make an error (1 mark). These supporting roles for human intervention in safety-critical systems will not disappear even if we automate complex systems (1 mark).

At the heart of the question is the fallacy that it is ever possible to engineer humans out of complex systems (1 mark). Even if we remove the operators' scope for intervention through automation, we do not remove the influence of technicians, engineers, managers and regulators. Good answers might link to other questions in the exam – for example arguing that error might arise in the application of development standards (1 mark) or refer to the question about configuration (1 mark). Bainbridge has also identified ironies of automation – the more we remove or restrict the operator scope for intervention then the less able we are to cope with potential error conditions (1 mark). Pilots who are not actively engaged in flying an aircraft can struggle to regain situation awareness when a flight management system is disengaged (1 mark).

Automation and resilience engineering are only two of the possible mitigations for human error. A number of other approaches are possible – answers might refer to human reliability analysis and the difficulty of quantifying the probability of human error. Others might refer to the use of training and of SOPs.