AVN-97/010

PJC/16-07-97

On Safety and Software Categorisation

A contribution to the report of

the study group on operational Computer Systems of the

UK Advisory Committee on the Safety on Nuclear Installations (ACSNI)

Software is a pervasive technology increasingly used in many different nuclear applications. Not all this software has the same criticality level with respect to safety. Thereby not all the software needs to be developed and assessed to the same degree of rigour. On the contrary, since development and V&V resources are always limited, attention in design and assessment should preferably be weighted to those parts of the system and to those technical issues that have the highest importance to safety.

The importance to safety of a computer based system and of its software is essentially determined by the functions it is required to perform in the plant, i.e. by its functional requirements. Non-functional requirements, such as reliability, are also constrained by the functionality. The importance to safety is therefore evaluated by a plant safety analysis with respect to the safety objectives and the design safety principles applicable to the plant. It is also determined by the consequences of the potential modes of failures of the computer system and of its software. The latter evaluation however is usually difficult because software failure occurrences are hard to predict.

The distinction between the safety and safety related classes - essential in the nuclear industry - is useful. Safety, Design and V&V requirements are in principle different for the two classes. In this respect, it is again interesting to distinguish the roles played by safety and reliability. The class to wich a system belongs is solely determined by the importance to safety of its functions, no matter how reliable the system is: a safety system that could be proven faultless with probability one still remains a safety system, and cannot be demoted to the safety related class.

There are however serious problems when these classes must be applied to software based systems. The difficulty comes from the impossibility of quantifying the "quality" of a piece of software, of guaranteeing a given level of quality, and of tailoring and controlling this quality by enforcing design and development procedures. Categorisation, therefore, cannot be used to define classes of "admissible software quality levels", nor to relax the requirements on the quality of the development and V&V processes for lower safety categories. The consequences of such relaxations on the quality of the software would in general be unpredictable. Quite surprisingly however, there are standards that allow certain relaxations of this kind, for instance requiring the use of formal methods for the highest criticality category and not for the next lower one.

However, these categories can be, and indeed are, advantageously used to relax the *reliability* constraints that are imposed on software based systems. Additional lines of defense - external to the software based system - can and should be used to reduce the importance to safety of these systems so that requirements in terms of reliability, availability and security can be lessened.

One is also faced with the problems raised by pieces of software which support functions of different criticality and which also must somehow interact, or communicate, or merely coexist on the same hardware. If one cannot prove that the less critical parts - whatever their behaviour is, correct or not - cannot adversely affect the more

critical ones, it is a common conservative practice to allocate the same highest critical level to all of them.

This problem, however, requires more attention. There is clearly a balance to be achieved here between the amount of design and V&V efforts that results from this conservative approach, and the amount that would be needed to obtain evidence that - despite possible interactions - the more critical functions cannot be affected by the behaviour of the lower critical parts.

Many computer system designs seem to ignore the possibility that the effort of the second kind may be reduced if the design is adequate. Separate processors coupled with one way simple proven protocols could be more advantageously used. It is also often forgotten that components can be isolated from one another not only physically but also logically. Logical firewalling can be spatial, e.g. through separate virtual memory spaces and protected memory segments, and/or temporal by enforcing appropriate time schedules protecting the more critical executions from overruns of the less critical ones.

Other logical mechanisms which need more investigation are suggested by the object model paradigm. Objects of different criticality levels could coexist in a system if mechanisms controlling the invocation and information flows and restricting the propagation of errors between distinct integrity levels are enforced. Advanced architecture designs of this kind are under development (e.g. the CE Esprit Guards project).