Software Safety - by Prescription or Argument?

by Tim Kelly (email: tpk@cs.york.ac.uk)

The topic for debate in the panel discussion was Software Safety Standards. The panel included industrial representation from aerospace (Praful Bhansali - Software Safety and Security Focal, Boeing), defence (Tony Cant - Defence Science and Technology Organisation) and the railways (Charles Page, Westinghouse Signals). The software safety research perspective was provided by Jim Welsh (from the Software Verification Research Centre, University of Queensland) and myself (representing Rolls-Royce plc and the High Integrity Systems Engineering Group at York). The discussion particularly focussed on the following questions:

- How do software safety standards differ from traditional safety standards?
- What is the role of the software safety case?

Addressing the first of these questions, the discussion highlighted the fact that there has been an intentional shift in the role of "traditional safety standards" away from prescription - telling the developer how to build their system safely - to target setting - forcing the developer to argue how they have achieved the appropriate targets. (This is certainly true of, for example, the U.K. Health and Safety Executive's approach to certification of the offshore and railways industries). However, with software safety standards (with both the U.K. Defence Standard 00-55 and IEC1508 cited as examples) we appear to be prescribing to a great level of detail what programming languages can be safely used, the sufficient level of testing, etc. Nowhere is this more apparent than with the ubiquitous Software Integrity Levels. These provide a perfect example of where (we assume) the standards-writers appear to have "done our thinking for us", e.g. concerning:

- How levels of integrity relate to target failure rates
- How integrity levels can be apportioned through system decomposition
- The appropriate development and assurance techniques for each integrity level
- The level of claim we can make having followed the integrity level guidelines

As no justification is provided of the requirements that emerge in the standards, it appears that the developer's role is to simply accept such things as givens. Although
some may say that this is within the nature of standards (i.e. to state, not explain) it was the opinion of the discussion that it can be too easy to comply with the standards whilst missing their (undisclosed) intent. Also, it was highlighted that this approach can present prohibitively harsh requirements for small-budget projects where perhaps, given the underlying rationale, requirements could justifiably be relaxed.

The answer? There appeared to be some consensus that where possible we should attempt to move away from prescription as our approach to ensuring software safety. Alternatively, more effort should be spent in determining the true, fundamental, requirements (targets) of software safety (for example, rather than specifying in detail the tools, techniques and languages of an "Integrity Level 4 process", instead working out the required attributes of a process to meet "Integrity Level 4"). The responsibility should then be placed on the software and system's developers to present an argument as to why their systems are safe - how they meet the fundamental intent of the standards. It was suggested that such an approach could provide benefit with respect to both safety - by making it harder to comply with standards in an "unthinking" manner - and financially by permitting arguments of appropriateness, and reduction of risks "As Low As Reasonably Practicable" (ALARP).

It would wrong to pretend that amongst all this "idealism" that there wasn't an underlying pragmatic concern from those representing industry. Specifically, they wanted standards to "tell them what to do" so that they didn't have to spend time and money determining what standards meant only in order to be told that they had got it wrong! It is was also recognised that with even with an ALARP approach, limits must be set on what is considered tolerable. The conclusion was therefore that although the current level of prescription may be too great, a careful balance must be maintained between the prescription and argument / target setting approaches.

The role of the software safety case

Arising out of discussion of the first question, it was clear that we had begun to answer the second - concerning the role of the software safety case. As with the role of "traditional" safety cases the purpose of the software safety case is very definitely to

*present a convincing, comprehensive and defensible argument that the system (in this case, the software) is acceptably safe to operate within a given operational context.*

Conversely, we appreciated that it is not the job of the software safety case to simply be the container for all the evidence required for compliance with the detailed requirements of the safety standards. Evidence of compliance alone does not a safety case make! In arguing the safety of our software components we must "get our hands dirty" with the details of the hazards of the systems in which they're placed. This means, first of all presenting an argument structured around sound and rigorous software hazard analyses (e.g. Software Hazards and Operability Studies as described by U.K Defence Standard 00-58). Secondly we must present, not in vague or ambiguous terms, but in a clear and specific manner, how the evidence available sufficiently addresses all of the software hazards identified. This reinforced Praful Bhansali's concern that we must be very clear how the effort spent in developing and
testing software has actually contributed to assuring safety. His challenge to anyone suggesting that he use a new safety analysis technique was that they should clearly explain why he needed it, i.e. where did it fit into the overall safety argument?

The question of the defensibility of the safety case was also raised by Dr Tony Cant. His experience of safety cases suggested that a problematic area was the use of quantitative models, particularly:

- The belief that a problem is well understood if it is quantified
- Insufficient "thought" and effort on those areas that had been quantified
- Unrealistic simplifications of a system in order to fit a model
- Inappropriate use of the figures contained within / resulting from a quantitative model

The conclusion arising from the resulting discussion was that the use of probabilities in the software safety case must always be fully justified and that although attractive (we all like numbers, don't we?) it was often an inferior substitute for well-structured qualitative reasoning.

Some Conclusions

The overriding feeling surrounding the various issues discussed was that building safety critical software is a uniquely difficult task. There are no easy answers - we should not look to standards wholly to prescribe an approach. We should be careful of over-simplification when translating qualitative problems (i.e. hazards) into the quantitative domain. Instead, our effort should be focussed on improving our capability to argue, lucidly and cogently, the safety case for our software systems.