KAPITEL 14

Software-Richtlinien für IEC 61508

Peter Ladkin 2010

14.1 Comments on Part 3, Annexes A and B

The method labels in the Annexes are a mix of overly-general with overly-specific. For example, „Formal Methods“ is a very general category, referring to the general use of mathematical and logical methods for assessing and designing software products. There are hundreds of formal methods, from methods of specification to methods of assessment of designs, of source code, and of machine code, code development methods, and derivation of test suites. In contrast, „Reliability Block Diagrams“ is a specific technique for assigning and calculating overall reliabilities of composite system parts given reliabilities of components, and assuming independent failures of those components.

Proposal 1: The table labels be annotated by commentary. Size of commentary would range from paragraphs (RBDs) to short essays (Formal Methods).

Proposal 1A: Rather than the commentary be inserted into the standard document Part 3, one could insert the commentary in a separate document or tool, such as the Uni Southampton/Uni Newcastle's Resilience Knowledge Base.

I understand that IEC 61508 is the IEC's second-biggest-selling document over some period of time and there are lots of industry sectors asking for guidance. At present, that guidance is not offered through between-revision activities of the national committees, but through various commercial, uncontrolled activities such as the training courses offered by various TÜV organisations (predominantly in Germany), Ron Bell's IET London Workshop on SIL determination, and the IET Cambridge course, both of the these latter predominantly HW-oriented and touching on SW methodology hardly at all. Specific guidance, that may be dynamically modified between revisions of the standard, is obviously needed, specifically on SW. The RKB arose out of the EU 6th Framework project ReSIST: Uni Newcastle was responsible for content, Uni Southampton for the engine. Were a utility such as the RKB to be used, there would likely need to be a formal agreement between owners and maintainers of the tool and the IEC. This could perhaps be accomplished through the EU 6th Framework program or its successor.

Proposal 2: The table entries refer to methods at the same general level of abstraction, performing similar tasks.

For example, „Reliability Block Diagrams“ is at the same „level“ as SW Fault Trees; both assess the reliability of composites from reliability data on components. It could be argued that source-code design at the level of state machines (Lustre) is at a similar level to SPARK annotation; neither are directly comparable to B specifications. Requirements specifications using Z are in principle comparable with requirements specifications using SPECTRM. Safety requirements specifications in OHA are comparable with safety requirements specifications in those two languages. Code generation from Lustre specifications (as in SCADE) is at the same level as code generation from rigorous-UML or from rigorous-Simulink. Compilation from SPARK source and compilation from MISRA C source are roughly comparable activities. Each particular category of methods has its own strengths and weaknesses, which need to be taken into account when using them. It makes sense to list the category-theoretic strengths and weaknesses. (I do not think it makes sense to attempt to list those of individual tools. The standard should not be working on the level of detail of individual tools.)

Proposal 2A: A classification of methods be used, similar to that suggested by example. The Annex table entries refer to the labels in that classification and recommendations, if thought wise, be made similarly.

14.2 A Proposal for Revised Requirements and Guidance on SW in Part 3

The current Part 3 recommends methods to be used in developing SW for a specified SW SIL level. The SW SILs are, according to the definitions, measures of the rate of dangerous failures caused by the software behavior. It is believed by many on the national committees that Part 3 sets no requirements on achievement, or demonstration of achievement, of specific SW SILs, primarily because well-known science shows it is not possible ex post facto to evaluate through testing any SW SIL requirement above SIL 1 (and in practice including SIL 1 also). Thus, the only requirement on SW consists of recommendations on methods to be used during the development process.

One may inquire about any demonstrated connection between use of specific methods as listed in Annexes A and B and the quality (in terms of dangerous failures per operational hour) of the resulting SW. Various specific companies such as Praxis HIS in the UK have instrumented their methodology over decades and can demonstrate delivered code quality, and improvement of that quality, over a period of time. Organisations such as Airbus and Eurocopter accumulate data on system behavior as a matter of safety routine, and metrics on code quality generated by, for example, use of SCADE are in principle available from this source. However, these metrics include as input not only the methods used, but a specific corporate culture as well as continuity in personnel. This is specifically true of CMMI-rated organisations, and it may be regarded as a weakness of CMMI that rated organisations do not deliver reliable metrics on quality achieved. I know of no reliable metrics for ultrahigh software quality (SIL 1 or higher) that depend only on the methods used.

The Part 3 requirements on SW development thus do not achieve the objective of SW SILs in the general case. Since organisations and culture are not part of the requirements, there is no demonstrated connection between the Part 3 requirements and the specified resulting SW quality. Because it must fail in this specific task of achieving SW SILs, and no other criteria for SW system properties are addressed, Part 3 does not specify any required demonstration of SW system properties at all. This situation is regarded by many (including me) as disastrous. It may be remedied, as follows.

Proposal A. An unambiguous, rigorous Functional Requirements Specification (FRS) be required. The FRS needs to be checkable for (i) consistency, and (ii) relative completeness. It be required that it is so checked and the methods and results to appear in the safety case.

Here, (i) is unproblematic in the current state of the art and state of the tools, but there are various ideas around what one means by (ii). (For example, Nancy Leveson has some ideas, and I have some orthogonal ones.) The idea of completeness is that there be assurance that all scenarios which can lead to hazards (by which I mean situations environment plus system state - in which there is a higher risk, not the 61508 sense of hazardous event = accident) have been accounted for in the FRS. Further, there be a requirement in the part of the standard which concerns operations that all hazards which are unaccounted for in the FRS be handled appropriately in operations (after-the-fact mitigation). (This particular requirement would entail a major change in industrial behavior. I personally know of various situations in which appropriate hazard analysis was not accomplished in light of operational discovery of hazards, and accidents ensued. One public example is the in-flight fire and subsequent destruction, with loss of life, of an RAF Nimrod reconnaissance aircraft in Afghanistan in 2006, but there were no SW issues identified in that case.)

Proposal B. (i) The SW Architecture/Design Spec be rigorous. (ii) There be a formal, rigorous, correct demonstration that the SWA/DS fulfils the FRS.

Proposal C. (i) If the SW is written using a higher-level programming language than machine code, there be defined a language to be called the Executable Source Code Level. (For C, or Modula, or Ada, or SPARK, source code would be the ESCL. For Java, one would have the option of specifying the ESCL as Java source, or as bytecode. For declarative languages such as Prolog, defining the ESCL might be a tricky. But the proposal is that there be one at some level.) (ii) The SW at the ESCL be rigorously, correctly demonstrated to fulfil the SWA/DS.

Proposal D. Compilation is defined to be an operation that translates ESCL into object code (OC) - the executable bytes that sit on the kit. Proposal: there be a rigorous, correct demonstration that the OC fulfils the ESCL.

Proposal E. There be a rigorous, correct demonstration that run-time errors do not cause or contribute to causing dangerous failures. One may do so by eliminating whole classes of them, as with SPARK, or by trapping and handling the raised exceptions.

Proposal Cluster F. Testing. Here, I think there is a bigger problem that has not yet been acknowledged. For, even though statistical testing cannot practically show the achievement of a SW SIL at the currently-defined levels, one does get some kind of assurance of SW fitness for purpose by even routine testing. The hard part the problem here is to make precise what kind of assurance this is, and specify how it is achieved.

Here a personal story and a moral. When I was writing mostly declarative code in the language REFINE, I wrote a time-interval calculation system over a few months. I did unit testing of the functions as I was writing them, performing (a) sanity checks, and, more thoroughly, (b) boundary-case calculations. Integration of the entire system took a programmer, who had no idea what I had written, two hours, mostly performing (a) and (b) at the integration level. She found one boundary case I had missed, I fixed it, and the code was on demo at the AAAI annual conference the next day. It has been used, I don't know how much, by the USAF in the project management part of its KBSA, and I have not heard of any errors. It is not a big system, but it would have been far, far more consumptive of time and effort to develop it in C. The point is that my testing was goal-directed, semantics-directed, and effective. Semantics-directed testing for (a) and (b) seem to me to be needed for any system. One could argue that they would be supplanted by effective SPARK-like formal methods; maybe so. But they or an equivalent are somehow needed; it is not the case that routine testing is without benefit for ultra-highly dependable systems.

There needs to be some kind of story on what unit testing and integration testing achieve and what is to be shown concerning that achievement. I don't have a well-formulated story at this point.

14.3 How the Proposals Will Achieve Higher SW Quality

The proposals enuciated above achieve a certain quality in SW developed by systematically excluding certain kinds of errors.

A rigorous FRS proved consistent avoids the error that various requirements contradict each other in subtle ways and no system can be built that satisfies the FRS.

A rigorous FRS proved complete, according to some notion of completeness, avoids requirements failures, in which an operational situation arises that was not envisaged and is not covered by the specification, and the system performs contrary to safety.

The requirements failures that will be avoided are those that would fall within the particular concept of completeness.

A rigorous, correct demonstration that SWA/DS fulfils FRS assures that the functional design of the SW will achieve the behavior specified in the FRS.

A rigorous, correct demonstration that ESCL code fulfils the SWA/DS shows that there are no coding errors.

A rigorous, correct demonstration that OC fulfils the ESCL shows that there are no compilation errors.

Using these proposals, SW will be delivered whose behavior is completely specified (according to some explicit notion of complete), and whose compiled code is guaranteed to fulfil the specification (except for certain explicit forms of run-time error which may not have been excluded during development, but which may be mitigated through system design).

Of course, the guarantees and demonstrations may be incorrect, but the levels of correctness achievable in assurance activities of this sort is much higher than the levels of correctness currently achieved in industrial code for safety critical systems (currently estimated by some who may be presumed to know at 1-3 errors per 1 KLOSC). For example, by making heavy, continual use of assurance during code development, Praxis HIS has demonstrated code quality of up to two orders of magnitude higher than the 1-3 errors per KLOSC just mentioned.

14.4 Software SILs

Proposal I: A series of SW SILs be developed that can be demonstrated by practical statistical testing to have been achieved by specific SW. The highest-level SIL be out of range (for example, pdfh lower than $O(10^{-5})$).

Proposal II: The SIL levels be defined through order-of-magnitude requirements, rather than the precise but arbitrary boundaries used at present

The SIL levels that could be defined according to this criterion, given that we talk about ultra-high dependability could start at $O(10^{-3})$.

Proposal III:

SW SIL 1 = pdfh of $O(10^{-3})$
SW SIL 2 = pdfh of $O(10^{-4})$
SW SIL 3 = pdfh of $O(10^{-5})$
SW SIL X = pdfh of $O(10^{-6})$ or lower (X indicates extreme)

Proposal IV: The achievement of SW SIL 1-3 be demonstrated through rigorous statistical testing. Evidence of SW SIL X be presented through (i) demonstrated achievement of SW SIL 3 through statistical testing, plus (ii) evidence of the quality (correctness) of the assurance activities carried out via Proposals A-E.

Using the exponential statistical model of software failure, which is the current state-of-the-art statistical model, Proposal IV involves statistical testing for the following periods of time:

- for 95% confidence level: 3 x pdfh
- for 99% confidence level: 4.6 x pdfh

The concept of dangerous failure means a failure which satisfies some external condition, namely that of being dangerous: the property of being dangerous or not is defined by the system in which the software is used, not by any internal property of the software. Thus in any statistical assessment of the software, it is failures which must be counted, since it is not possible to distinguish dangerous failures from failures with respect to the software alone.

Combining Proposals III and IV (i), and using the exponential statistical model of software failure, along with the remark that it is failures which count for the purpose of statistical assessment, the periods of time required for statistical testing without an observed failure are:

- For SW SIL 1:
  - 3,000 hours of statistical testing without failure for 95% confidence
  - 4,600 hours of statistical testing without failure for 99% confidence
- For SW SIL 2:
  - 30,000 hours ($\approx$ 3 years, 5 months) of statistical testing without failure for 95% confidence
  - 46,000 hours ($\approx$ 5 years, 3 months) of statistical testing without failure for 99% confidence
- For SW SIL 3:

  - 300,000 hours ($\approx$ 34 years, 3 months) of statistical testing without failure for 95% confidence
  - 460,000 hours ($\approx$ 52 years, 6 months) of statistical testing without failure for 99% confidence
- For SW SIL X: a requirement for pdfh of $O(10^{-6})$ would require
  - 3,000,000 hours ($\approx$ 342 years 5 months) of statistical testing without failure for 95% confidence
  - 4,600,000 hours ($\approx$ 525 years 1 month) of statistical testing without failure for 99% confidence

Concerning Proposal IV (ii), I have not yet formulated a proposal as to what evidence is appropriate.

14.5 Applicable Techniques and Recommendations (Annex A)

There is some degree of consensus amongst internationally-recognised experts on critical software that the current Tables in Annex A are vague. There is also consensus that merely following the recommendations in the Tables in Annex A does not provide any demonstrable correlation with either assured properties or assured quality of the resulting SW. I propose a list of methods which are applied and for which various tools exist, which is fine-grained enough to enable some properties of the resulting software to be meaningfully asserted, provided the methods have been appropriately applied.

Proposal AA-1: Instead of the entries semi-formal methods and formal methods, including ... in the Tables in Annex A, I propose one single entry, Formal Methods, to be HR at all SIL levels. In Annex B, I propose a supplementary table for Formal Methods, to consist of the following entries:

1. Formal functional requirements specification (FRS)
2. Formal FRS analysis
3. Formal safety requirements specification (FSRS)
4. Formal FSRS analysis
5. Automated proving/proof checking of properties (consistency, completeness of certain types) of FRS and FSRS

6. Formal modelling, model checking, and model exploration of FRS, FSRS
7. Formal design specification (FDS)
8. Formal analysis of FDS
9. Automated proving/proof checking of fulfilment of the FRS/ FSRS by FDS
10. Formal modelling, model checking, and model exploration of FDS
11. Formal determininistic static analysis of FDS (information flow, data flow, possibilities of run-time error)
12. Codevelopment of FDS with ESCL
13. Automated source-code generation from FDS or intermediate specification (IS)
14. Automated proving/proof checking of fulfilment of FDS by IS
15. Automated verification-condition generation from/with ESCL
16. Rigorous semantics of ESCL
17. Automated ESCL-level proving /proof checking of properties (such as freedom from susceptibility to certain kinds of run-time error)
18. Automated proving/proof checking of fulfilment of FDS by ESCL
19. Formal test generation from FRS
20. Formal test generation from FSRS
21. Formal test generation from FDS
22. Formal test generation from IS
23. Formal test generation from ESCL
24. Formal coding-standards analysis (SPARK, MISRA C, etc)
25. Worst-Case Execution Time (WCET) analysis
26. Monitor synthesis/runtime verification

Proposal AA-2: It be required in the safety case to state which of these techniques was applied, which tools were used in that application, and what properties of the SW were achieved through use of the technique.

Proposal AA-3: It be required in the safety case to demonstrate that the object code fulfils the FRS.

For (hypothetical) example:

Technique 9, automated proving of fulfilment of FSRS by FDS was undertaken. The method used was TLA. The FSRS is expressed in Z: the FDS in TLA+. Therefore a translation of the Z in FSRS into TLA+ was undertaken manually (see accompanying documentation). The resulting TLA+ specification was shown to be machine-closed. TLA was used manually to prove refinement of the TLA+ translation of FSRS by FDS. Since the task involved FSRS only, only proof of refinement of the safety properties was undertaken. The Merz TLA prover in Isabelle was used to proof-check the manual proof. Three errors were found (see accompanying documentation), corrected (ditto) and the correction proof-checked using the Merz prover.