Assurance Points in Software Development

Peter Bernard Ladkin

Causalis Ingenieurgesellschaft mbH/Causalis Limited

21 May 2018

Clarification

The original slideset was designed as accompaniment to the talk.
This slideset is a modified version for reading.

First, a prolegomenon

Then the slides used in the talk

Prolegomenon

- Why standards are important
- the Basic Safety Standard IEC 61508
  - what is in it concerning cybersecurity
  - that cybersecurity very possibly "changes the game"
- disagreements about the efficacy of so-called "formal methods" (FM)
- examples of inconsistency in critical-system functional specifications, courtesy of Michael Jackson
- from safe.tech 2018: critical software developed painstakingly from ..... "User Stories"!
- unravelling "safety requirements" and "software safety requirements" in IEC 61508
  - not for the faint-hearted
  - but FM is "highly recommended" for the more rigorous levels

RVS Rechnernetze und Verteilte Systeme

CAUSALIS

Prolegomenon, cont'd

- A list of the 26 points in SW development at which objective properties of SW and it documentation can be assured with the help of FM (and, often, not without such help)
- details on the current IEC New Work Item Proposal 65A/867/NP up for voting
- key questions motivating this work
  - should we be building critical systems which we do not guarantee are fit for purpose?
  - does it still "cost too much" for such a guarantee?
- dependable software... does not do what we want to avoid
- but (IEC 61508-3 subclause 7.7.2.7 a)) "testing shall be the main validation method..."
  - Rubbish! – it can't be, and the proof is simple
  - this has been known and remarked for 49 years

Prolegomenon, cont'd

- for example, let us consider what needs to be analysed concerning functional requirements
  - we have seen issues with consistency
  - what about completeness?
  - Lutz 1993: safety-related errors arose most frequently ($> 98\%$!) from discrepancies between requirements specifications and conditions encountered in operation
  - completeness was thus the biggest problem
  - anecdotes suggest that matters have not changed very much
  - .... because, we might suppose, the lesson was not well learned

Prolegomenon, cont'd

- there is a problem here with IEC 61508 and its concepts of verification and validation
- V & V as a folk saying
  - verification is assuring "you have built the system right"
  - validation is assuring "you have build the right system"
  - validation in this conception addresses the Lutz phenomenon
- hard hats on for the convoluted definitions of V & V from IEC 61508
  - astonishingly, they are almost the same
  - they don't appear to address the Lutz phenomenon
- we can do better – to finish, a simple "Waterfall" scheme motivating the 26 assurance points
- Thanks to the German Federal Ministry for Economic Affairs and Energy for support of some of this work

Now for the talk slides

Standards

- Claim: standards are important
  - if two people communicate, electronics on both ends needs to have the same transformation of speech into electrical/electronic signals, and the same behaviour concerning how those signals are handled
    - Morse Code
    - the Internet
  - to recharge your electric car at the end of a journey, then your connections & connectors need to be compatible, and to have compatible behaviour, with those at the end of your journey
- Methodological standards are also important
  - industrial safety-critical equipment is often available as components, which are then assembled into a larger system
    - assessors have similar overall information on each of the components
    - and how they contribute to key properties of the end system
  - major suppliers supply internationally – it is convenient, and less expensive, to conform to one standard rather than many

IEC 61508-3:2010

- Current "Basic Safety Standard" (generic standard) for software development in safety-critical applications
    - does not apply to aerospace
    - does not apply to medical systems
    - required for industrial-process plant (IEC 61511 refers to it)
    - "adapted" for rail control and protection systems (EN 50128)
    - "adapted" for road vehicles (ISO 26262)

Cybersecurity?

- Current events include penetration of various critical systems
  - either with hostile intent; e.g., electricity-supply systems in Ukraine
  - or with apparently-criminal intent; e.g., Equifax data leak
  - or as proof of concept; e.g., remote control of a Jeep (Miller/Valasek)

Cybersecurity in IEC 61508-1 (General System)

“*If the hazard analysis identifies that malevolent or unauthorised action, constituting a security threat, as being reasonably foreseeable, then a* **security threats analysis** *should be carried out.*
NOTE 1 For reasonably foreseeable misuse see 3.1.14 of IEC 61508-4.
.....
NOTE 3 For guidance on **security risks analysis***, see IEC 62443 series.*
NOTE 4 Malevolent or unauthorised action covers security threats.”

“Subclause 7.5.2.2
If security threats have been identified, then a **vulnerability analysis** *should be undertaken in order to specify security requirements.*
NOTE Guidance is given in IEC 62443 series.

Cybersecurity in IEC 61508, II

- none of these terms are defined
- notice that nothing says you have to do anything. Just analyse
- next edition of IEC 61508? 2020 earliest validity date.

Cybersecurity in IEC 61511

- There is a little more in IEC 61511 (process-plant systems)
- mainly concerning access control to systems
- about two pages total
- the current standard is brand-new.

Why Do I Mention Cybersecurity So Early?

It is different from safety vulnerabilities

- With safety, it is nature/happenstance presenting your system with unwelcome inputs
  - if those inputs occur rarely, they will likely continue to occur rarely
  - frequency is plausibly stochastic
- whereas in cybersecurity, unwelcome inputs are devised intentionally by a more or less expert human (or machine)
  - those inputs occurred rarely, even not at all (zero day)
  - then they can start occurring as fast as an attacking machine can generate and input them
  - the frequency is not at all stochastic; it is never, then often

Colleagues have posited that cybersecurity concerns are a game-changer for rigorous assurance

Exercise: Spot the Cybersecurity Vulnerability

Some Human Conflicts

- "Formal methods don't work!" (reputed: B. Boehm, 1980's)
    - Many, including top computer scientists and some companies, reply: "yes, they do, depending specifically on what you want to achieve"
    - A twofold response to this reply:
        - ★ "But we can't learn them"
        - ★ "It costs too much (people, time) for the benefit"
- This is still a common twofold response
    - 6.5 years after the first version of this slide
    - 8 years after I began this movement in formal standardisation work!
- Meanwhile, Altran UK can discharge >150,000 proof obligations on 0.25M LOC ATM SW in 15 minutes on a desktop computer
    - they discharge the proof obligations for the dependability case on a daily basis after daily updates

Examples from System Functional Requirements via Michael Jackson

Example I: Mode logic of a Flight Guidance System
Miller, Tribble, Whalen, Heimdahl, 2006

- Original requirements
  - expressed as "*shall*" statements in natural language
  - formalised by the authors in CTL
  - checked against a large formal model of the intended system behaviour
- Original requirements
  - complete?
  - unambiguous?
  - consistent?
- "*inevitably, they were not*" (M. Jackson, Festschrift Glinz, 2012)

System Functional Requirements Example I (Miller et al.)

"*If Heading Select mode is not selected, Heading Select mode shall be selected when the HDG switch is pressed on the Flight Control Panel.*"

- Requirement "*well-validated and non-controversial*"
- Checking showed two errors
  - the event of pressing the HDG switch could be pre-empted by a simultaneous event of higher priority
  - the event of pressing the HDG switch should be ignored unless the side on which it occurs is active

That was 12 years ago. Are we better today?

System Functional Requirements
Example II (Harel)

- Chemical-manufacturing plant
- Specification included, in three totally different locations,
  - *"If the system sends a signal HOT then send a message to the operator"*
  - *"If the system sends a signal HOT with* $T > 60\,\text{deg}$ *then send a message to the operator"*
  - *"When the temperature is maximum, the system should display a message on the screen, unless no operator is on the site except when* $T < 60\,\text{deg}$*"*
- (Partial solution: Statecharts)

That was even longer ago. Are we better today?

The State of the Art (safe.tech Conference, April 2018)

- Engineer from a well-regarded critical-software company
  - active in automotive, avionics, and so on
- “agile” development applied to critical software
- very fine-grained
  - meticulous version control
  - multiple builds per team per day
  - multiple inspections per build
- impressive: reminder that “traditional” Waterfall/V-Model concerns documentation style and not the means of coding
- PBL: “*What about the functional requirements*”
- reply: “*..... User Stories ......*”
  - not even Use Cases! (Structured User Stories?)

What is Actually Said in IEC 61508-3

IEC 61508-3:2010 subclause 7.2.2 Note 2:

> *For the selection of appropriate techniques and measures . . . , the following properties . . . of the software safety requirements specification should be considered:*
>
> – *completeness with respect to the safety needs to be addressed by software;*
>
> – *correctness with respect to the safety needs to be addressed . . .*
>
> – *freedom from intrinsic specification faults, including . . . ambiguity*
>
> – *understandability . . .*
>
> – *freedom from adverse interference [from] non-safety functions . . .*
>
> – *capability of providing a basis for verification and validation*

What is a Software Safety Requirements Specification?

- "*software safety requirements*" undefined in IEC 61508-4 (the Definitions part)
- IEC 61508-1:2010 subclause 7.5 Overall Safety Requirements
  - *7.5.2.1 A set of all necessary overall safety functions shall be developed based on the hazardous events derived from the hazard and risk analysis. This shall constitute the specification for the overall safety functions requirements.*
  - *7.5.2.2 If security threats have been identified, then a vulnerability analysis should be undertaken in order to specify security requirements.*
  - *7.5.2.3 For each overall safety function, a target safety integrity requirement shall be determined that will result in the tolerable risk being met. . . . This shall constitute the specification for the overall safety integrity requirements.*

Safety Requirements in IEC 61508: Summary

- Safety requirements consist of
  - what safety functions there are to be (safety functions requirements)
  - ??? security requirements ??? (needs work!)
  - a target safety integrity requirement for each safety function
  - plus some more detail (IEC 61508-1:2010 subclause 7.10 says what)
- A "*safety integrity requirement*" is specified in IEC 61508-4 subclause 3.5.8 (the definition of) *safety integrity level (SIL)*, Note 2:
  - *Safety integrity levels are used for specifying the safety integrity requirements of the safety functions to be allocated to the E/E/PE safety-related systems.*
- a SIL is a reliability requirement on a safety function (four levels: SIL 1 to SIL 4)

Software Safety Requirements

- SW safety requirements consist of (IEC 61508-3:2010 subclause 7.2.2.10)
  - safety functions to be implemented in SW
  - the "*software systematic capability*"
- *systematic capability* is (Part 4, subclause 3.5.9)

  measure ... of the confidence that the systematic safety integrity of an element meets the requirements of the specified SIL, in respect of the specified element safety function, ...
- "*element*" is not a well-understood concept
- PBL: SW has to run on HW, and it is this SW+HW which usually/often/sometimes/occasionally constitutes the "element"

Software Safety Requirements in IEC 61508: Summary

- software is part of an element
  - which implements a safety function
  - which has a SIL, namely, a reliability requirement
- software safety requirements specify
  - the safety function, and
  - your confidence with which the safety function attains its SIL, namely, your confidence that a certain reliability is attained
- that sounds as if numerical (statistical) evaluation is necessary
- quantification is "recognised" to be generally inappropriate for software, so this turns out to mean a certain level of care in software development (accompanied by the waving of many hands)
  - "formal methods" are "highly recommended" for SIL 3 and SIL 4 functions (so much for "Formal methods don't work"!)
  - any kind of guidance in their use is lacking
  - many SW houses balk at using them
    - ★ ?in favour of "User Stories"?

Points at Which Rigorous Assurance can be Applied in Safety-Critical Software Development

- 26 points at which objective properties of SW and its documentation can be established using mature mathematical/logical methods
  - Bishop, Bloomfield, Knight, Ladkin, Littlewood, Rushby, Strigini, Thomas: the "Finsbury Group", 2009-2011
- I call these **assurance points**
  - some are objects
    - e.g., a functional requirements specification
  - some are arguments showing properties and relations between these objects
    - e.g., that the design spec fulfils the functional requirements spec
    - "fulfils" is a binary relation between two specifications

The 26 Points, 2011 Version

- 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)

RVS Rechnernetze und Verteilte Systeme

CAUSALIS

The 26 Points, continued

- 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

⟶ IEC New Work Item Proposal

- Work presented January 2010 to DKE AK 914.0.3
  - "Safe Software" Working Group of the German National Committee (DNC) responsible for IEC 61508
- Final proposal (not all 26 assurance points) agreed October 2016
- DNC resolved November 2017 to propose it to the IEC
- forwarded to the IEC in March 2018
- IEC project 65A/867/NP currently in vote from country participants in SC65A (31 countries) until June 2018
  - *Requirements and Guidance in the use of mathematical and logical techniques for establishing exact properties of software and its documentation*

Key Questions Motivating this Work

- Moral question, then as now:

  should we any longer be building systems which we don't guarantee are fit for purpose?

- Business/social question:

  does it (still) "cost too much for the benefit" to build systems with some kind of guarantee they are fit for purpose?

- Engineering/economics question: why is this needed?

Down to Brass Tacks

- Dependable SW does what we want it to do, as far as it can
  - Do we *really* know what we want?
  - How do we tell that we know what we want?
  - How do we tell the SW does it?
- But also **it doesn't do what we don't want it to do**
  - How do we know what we don't want?
  - How do we assure ourselves that we know?
  - How do we tell that the SW doesn't do any of that?

What IEC 61508-3 Says About This

“7.7.2.7
The validation of safety-related software aspects of system safety shall meet the following requirements:
a) **testing shall be the main validation method for software***; analysis, animation and modelling may be used to supplement the validation activities;*
b) the software shall be exercised by simulation of:.....”

Testing as the "Main Validation Method"

Much of the documentation requirements on software in IEC 61508-3:2010 consists of test plans and what shall be done when tests fail (about 1/3 out of almost 60 pieces of documentation)

But it has been well-expressed for 49 years that

- *Testing shows the presence, not the absence of bugs* (EWD 1969)
  - and it is the *absence* of unsafe behaviour which we are trying to demonstrate/argue
- *Testing for phenomena that are not expected to occur in the system lifetime takes 3 times the system lifetime for 95% confidence and 4.6 times the system lifetime for 99% confidence (Littlewood-Strigini 1993, J. Bernoulli 1713 (posth.), S. Poisson 1837)*
  - which is clearly impossible

The Core of an Ideal Safety Standard

- Determine what we don't want the system to do (Accidents)
  - as completely as possible
  - provide the assurance that we have everything
- determine how it could happen (Hazards)
  - as completely as possible
  - if we go too far, that's OK
  - provide the assurance of coverage (completeness)
- "apportion" the hazard behavior to the system components (including software)
  - show/demonstrate/prove the apportionment covers the hazards
  - avoid or mitigate each hazard
- for the SW, indeed any component: repeat the above

Evaluating the Core

To illustrate what follows from this:

Requirements

- they should be specified
- they should be consistent
  - if they are not consistent, they cannot be fulfilled
  - because different domains in a complex system have different expertise, it is not only common but easy to generate conflicting requirements from different domains (e.g., the examples reported by M. Jackson)
- they should be complete ....

Evaluating the Core, continued

Requirements, continued

- ....
- they should be complete
  - one meaning of "complete": specifying exactly which system behaviours are acceptable and which not
  - another meaning: considering **all** the circumstances in which the system will operate, and specifying its behaviour in all these circumstances
  - a third meaning: define a "view", that is, use specific, limited vocabulary to describe system+environment behaviour; show formal completeness (first meaning) with respect to this view; consider many views.

Completeness of Requirements

Requirements completeness, second meaning: considering all the circumstances in which the system will operate, and specifying its behaviour in all these circumstances

The second meaning is

- a usual, practical meaning
  - it is often artificial to limit the system vocabulary in a way that would enable one to address the first meaning
- the source of most complex critical-system failures
  - Evident since at least Lutz, 1993: "*Safety related software errors are shown to arise most commonly from discrepancies between the documented requirements specifications and the requirements needed for correct functioning of the system and misunderstandings of the software's interface with the rest of the system*"

Definitions (Colloquial)

- verification
  - you have built the system right
    - ★ machine code fulfils source-code semantics fulfils design fulfils functional requirements
- validation
  - you have built the right system
    - ★ functional requirements are complete . . .
    - ★ that is, all operational circumstances are considered
    - ★ absence of the Lutz phenomenon, as far as possible
    - ★ . . . and functional requirements are correct (Jackson examples)
- (occasionally the other way round)

IEC 61508 Definition I

“3.8.1
verification
confirmation by examination and provision of objective evidence that the requirements have been fulfilled

NOTE In the context of this standard, verification is the activity of demonstrating for each phase of the relevant safety lifecycle (overall, E/E/PE system and software), **by analysis, mathematical reasoning and/or tests**, *that, for the specific inputs, the outputs meet in all respects the objectives and requirements set for the specific phase.*

IEC 61508 Definition I, continued

"*EXAMPLE Verification activities include*

- *reviews on outputs (documents from all phases of the safety lifecycle) to ensure compliance with the objectives and requirements of the phase, taking into account the specific inputs to that phase;*
- *design reviews;*
- *tests performed on the designed products to ensure that they perform according to their specification;*
- *integration tests performed .... and by the performance of environmental tests.....*"

?? Where is any suggestion of Static Analysis ??

?? Where is any suggestion that verification activities persist into operational maintenance??

IEC 61508 Definitions II

"3.8.2
validation
confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use *are fulfilled*

.....

NOTE 2 Validation is the activity of demonstrating that the safety-related system under consideration, before or after installation, meets in all respects the safety requirements specification for that safety-related system. Therefore, for example, software validation means confirming by examination and provision of objective evidence that the software satisfies the software safety requirements specification."

Exercise: spot the difference.

Apropos Completeness of Requirements

And

3.8.1
verification
confirmation by examination and provision of objective evidence that the requirements have been fulfilled

3.8.2
validation
confirmation by examination and provision of objective evidence that the particular requirements for a specific intended use *are fulfilled*

- Neither of these address the Lutz phenomenon, the discrepancy between operational circumstances and requirements specification
- And this in 2018, a quarter century after Lutz's report

We Can Do Better

- Waterfall (modified-Royce): SW+doc has four “life stages”
- these concern, really, the organisation of the documentation
  - an engineered system is more than the physical construction – *the purpose* must be given
  - if we are to speak of dependability, it must be argued that *the purpose is fulfilled*
  - this argument constitutes the (functional) documentation
- The stages
  - Functional Requirements development and specification
  - Design
  - “Source” code generation: more generally, intermediate code objects
  - Object code (linked and loaded)
- In each of these stages, it is possible using mature mathematical methods to assess objective properties of the software and its documentation. Let us see where.
- (Yes, I have left out testing and maintenance)

Assessment Points I

- Functional Requirements
  - formal specification, suitable for checking consistency
  - check consistency
  - analyse for completeness
    - use restricted language, check first-meaning completeness
    - model checking
    - formal operational-model building and exploration
- Design
  - formal specification
    - check for consistency (in many cases assured by default)
    - check for implementability: e.g., information-flow analysis
- Compare design against requirements
  - does the formal design specification implement the formal requirements specification?
    - discrepant formal languages plus interpretation, refinement

Assessment Points II

- Generate source code
  - does the source code refine the formal design specification?
  - key: supply-chain issues
    - Development environment
    - Libraries
- Iterate for as many stages as one has intermediate code
  - Java "source"
  - Java byte code
- Generate object code
  - does the object code refine the source code?
    - How well is compiler/link/load sequence understood?
- Consider
  - coding-standards analysis
  - WCET analysis
  - run-time monitoring

Testing?

I have not addressed testing. Despite its “primacy” for “validation”.

Another time, maybe........

Keep cycling! And recycling!

Re Cycling: the Vulnerability Not Exploited

RVS Rechnernetze und Verteilte Systeme

CAUSALIS

Acknowledgement

We gratefully acknowledge the support of the German Federal Ministry for Economic Affairs and Energy (BMWi) in Grants 03TNG006A and 03TNG006B, Project Harbsafe, for our work on issues with safety and security terminology in standards

Finish

All done

Fertig

Th th thth-at's all, folks!

Thanks Fini

so much

Danke

Merci !