Issues in Safety Assurance

Martyn Thomas

martyn@thomas-associates.co.uk

Summary

I want you to agree that:

- Safety Integrity Levels are harmful to safety and should be abandoned.
- We must urgently design a new basis for developing and assuring/certifying software-based safety systems.

Safety-Related Systems

Computer-based safety-related systems *(safety systems):*

- sensors, actuators, control logic, protection logic, humans …
- typically, perhaps, a few million transistors and some hundreds of kilobytes of program code and data. And some people.
- *Complex.*
- *Human error is affected by system design. The humans are* part *of the system.*

Why systems fail:
some combination of ...

- inadequate specifications
- hardware or software design error
- hardware component breakdown (*eg thermal stress*)
- deliberate or accidental external interference (*eg vandalism*)
- deliberate or accidental errors in fixed data (*eg wrong units*)
- accidental errors in variable data (*eg pilot error in selecting angle of descent, rather than rate*)
- deliberate errors in variable data (*eg spoofed movement authority)*
- human error (*eg shutting down the wrong engine)*
- *...... others?*

Safety Assurance

Safety Assurance should be about achieving *justified* confidence that the frequency of accidents will be acceptable.

- **Not** about satisfying standards or contracts
- **Not** about meeting specifications
- **Not** about subsystems

… but about whole systems and the probability that they will cause injury

So ALL these classes of failure are our responsibility.

Failure and meeting specifications

A system **failure** occurs when the delivered service deviates from fulfilling the system **function**, the latter being what the system *is aimed at*. (J.C Laprie, 1995)

The phrase “what the system is aimed at” is a means of avoiding reference to a system “specification” - since it is not unusual for a system’s lack of dependability to be due to inadequacies in its documented specification.

(B Randell, Turing Lecture 2000)

The scope of a safety system:

The developers of a safety system should be accountable for all possible failures of the physical system it controls or protects, other than those explicitly excluded by the agreed specification.

Estimating failure probability

from various causes

- ✘ Inadequate specifications
- ✘ hardware or software design error
- ✓ hardware component breakdown (*component data*)
- ✘ deliberate or accidental external interference
- ✘ deliberate or accidental errors in fixed data
- ✓ accidental errors in variable data/human error (*HCI testing and psychological data*)
- ✘ deliberate errors in variable data

➔ *System failure probabilities cannot usually be determined from consideration of these factors.*

Assessing whole systems

In principle, a system can be monitored under typical operational conditions for long enough to determine *any* required probability of unsafe failure, from any cause, with *any* required level of confidence.

In practice, this is rarely attempted. Even heroic amounts of testing are unlikely to demonstrate better than 10^{-4}/ hr at 99%.

So what are we doing requiring 10^{-8}/hr (and claiming to have evidence that it has been achieved?).

I believe that we need to stop requiring/making such claims.

… so let's look at SILs

Safety Integrity Levels

Low Demand: < 1/yr AND < 2* proof-test freq.

Safety integrity level	**Low demand mode of operation** (Average probability of failure to perform its design function on demand)
4	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-2}$ to $< 10^{-1}$

IEC 61508

Proof testing is generally infeasible for software functions.

Why should a rarely-used function, frequently re-tested exhaustively, and only needing 10^{-5} pfd, have the same SIL as a constantly challenged, never tested exhaustively, 10^{-9}pfh function? *Low demand mode should be dropped for software.*

Safety Integrity Levels
High demand

Safety integrity level	**High demand or continuous mode of operation** (Probability of a dangerous failure per hour)
4	$\geq 10^{-9}$ to $< 10^{-8}$
3	$\geq 10^{-8}$ to $< 10^{-7}$
2	$\geq 10^{-7}$ to $< 10^{-6}$
1	$\geq 10^{-6}$ to $< 10^{-5}$

IEC 61508

Even SIL 1 is beyond reasonable assurance by testing.

IEC 61508 recognises the difficulties for assurance, but has chosen to work within current approaches by regulators and industry.

What sense does it make to attempt to distinguish single factors of 10 in this way? Do we really know **so much** *about the effect of different development methods on product failure rates?*

How do SILs affect software?

- SILs are used to recommend software development (including assurance) methods
  - stronger methods more highly recommended at higher SILs than at lower SILs
- This implies
  - the recommended methods lead to fewer failures
  - their cost cannot be justified at lower SILs

Are these assumptions true?

(1) SILs and code anomalies

(source: German & Mooney, Proc 9th SCS Symposium, Bristol 2001)

- Static analysis of avionics code:
  - software developed to levels A or B of DO-178b
  - software written in C, Lucol, Ada and SPARK
  - residual anomaly rates ranged from
    - 1 defect in 6 to 60 lines of C
    - 1 defect in 250 lines of SPARK
  - 1% of anomalies judged to have safety implications
  - **no significant difference between levels A & B.**
- *Higher SIL practices did not affect the defect rates.*

Safety anomalies found by static analysis in DO 178B level A/B code:

- Erroneous signal de-activation.
- Data not sent or lost
- Inadequate defensive programming with respected to untrusted input data
- Warnings not sent
- Display of misleading data
- Stale values inconsistently treated
- Undefined array, local data and output parameters

-Incorrect data message formats
-Ambiguous variable process update
-Incorrect initialisation of variables
-Inadequate RAM test
-Indefinite timeouts after test failure
-RAM corruption
-Timing issues - systems runs backwards
-Process does not disengage when required
-Switches not operated when required
-System does not close down after failure
-Safety check not conducted within a suitable time frame
-Use of exception handling and continuous resets
-Invalid aircraft transition states used
-Incorrect aircraft direction data
-Incorrect Magic numbers used
-Reliance on a single bit to prevent erroneous operation

Source: Andy German, Qinetiq. Personal communication.

(2) Does strong software engineering cost more?

- Dijkstra's observation: avoiding errors makes software cheaper. (Turing Award lecture, 1972)
- Several projects have shown that very much lower defect rates can be achieved alongside cost savings.
  - (see http://www.sparkada.com/industrial)
- *Strong methods do not have to be reserved for higher SILs*

SILs: Conclusions

- SILs are unhelpful to software developers:
  - SIL 1 target failure rates are already beyond practical verification.
  - SILs 1-4 subdivide a problem space where little distinction is sensible between development and assurance methods.
  - There is little evidence that many recommended methods reduce failure rates
  - There is evidence that the methods that *do* reduce defect rates also save money: *they should be used at any SIL*.

SILs: Conclusions (2)

- SILs set developers impossible targets
  - so the focus shifts from achieving adequate safety to meeting the recommendations of the standard.
  - this is a shift from *product* properties to *process* properties.
  - but there is little correlation between process properties and safety!
- So SILs actually damage safety.

A pragmatic approach to safety

- Revise upwards target failure probabilities
  - current targets are rarely achieved (it seems) but most failures do not cause accidents
  - … so current pfh targets are unnecessarily low
  - safety cases are damaged because they have to claim probabilities for which no adequate evidence can exist - so engineers aim at satisfying standards instead of improving safety
- We should press for current targets to be reassessed.

A pragmatic approach to safety (2)

- Require that *every* safety system has a formal specification
  - this inexpensive step has been shown to resolve many ambiguities
- Abandon SILs
  - the whole idea of SILs is based on the false assumption that stronger development methods cost more to deploy. Define a core set of system properties that must be demonstrated for all safety systems.

A pragmatic approach to safety (3)

- Require the use of a programming language that has a formal definition and a static analysis toolset.
  - A computer program is a mathematically formal object. It is essential that it has a single, defined meaning and that the absence of major classes of defects has been demonstrated.

A pragmatic approach to safety (4)

- Safety cases should start from the position that *the only acceptable evidence* that a system meets a safety requirement is an independently reviewed proof or statistically valid testing.
  - Any compromise from this position should be explicit, and agreed with major stakeholders.
  - This agreement should explicitly allocate liability if there is a resultant accident.

A pragmatic approach to safety (5)

- If early operational use provides evidence that contradicts assumptions in the safety case (for example,if the rate of demands on a protection system is much higher than expected), the system should be withdrawn and re-assessed before being recommissioned.
  - This threat keeps safety-case writers honest.

A pragmatic approach to safety (6)

- Where a system is modified, its whole safety assessment must be repeated *except* to the extent that it can be proved to be unnecessary.
  - Maintenance is likely to be a serious vulnerability in many systems currently in use.

A pragmatic approach to safety (6)

- COTS components should conform to the above principles
  - Where COTS components are selected *without* a formal proof or statistical evidence that they meet the safety requirements in their new operational environment, the organisation that selected the component should have strict liability for any consequent accident.
  - “proven in use” should be withdrawn.

A pragmatic approach to safety (7)

- All safety systems should be warranted free of defects by the developers.
  - The developers need to “keep some skin in the game”
- Any safety system that could affect the public should have its development and operational history maintained in escrow, for access by independent accident investigators.

Safety and the Law

- In the UK, the Health & Safety at Work Act's ALARP principle creates a legal obligation to reduce risks as low as reasonably practicable.
- Court definition of *reasonably practicable:* "the cost of undertaking the action is not grossly disproportionate to the benefit gained."
- In my opinion, my proposals *would reduce risks* below current levels and *are reasonably practicable.* Are they therefore legally required?

Summary

- Safety Integrity Levels are harmful to safety and should be abandoned.
- We must urgently design a new basis for developing and assuring/certifying software-based safety systems.

Do you agree?