21
Critical Systems Engineering

Distributed Embedded Systems

Philip Koopman

November 18, 2015

Carnegie Mellon

Why Is Automotive SW Safety A Big Deal?

- **Mechanical components break all the time**
  - And we expect that to be normal
  - BUT, computers are expected to have “perfect” functionality/software

- **Possible reasons for demands that software be perfect**
  - Software defects are design defects, and we don’t tolerate mechanical design defects either
  - Software failure modes can be very non-intuitive or seemingly random, causing driver to feel lack of control over situation
  - Redundancy might not work (both SW copies can break the same way)

- **Can a software feature be just as unreliable as a mechanical feature?**
  - Possibly not if failures are due to software/design defects rather than wearout
  - Worse, software might provide extra safety features that drivers rely upon, causing accidents if feature is inoperative
  - Therefore, it might be that a software version has to be safer than a mechanical version to withstand public scrutiny

Automotive Software Woes – 1

Source: http://www.leshatton.org/Documents/Hatton_Xmas1_03.pdf

- **In July 1999, General Motors has to recall 3.5 million vehicles because of an anti-lock braking software defect.**
  - Stopping distances were extended by 15- 20 meters. Federal investigators received reports of 2,111 crashes and 293 injuries.
  - http://autopedia.com/html/Recall_GM072199.html

- **In September 2000**, Production of year 2001 models of Ford Windstar, Crown Victoria, Mercury Grand and Lincoln stopped because of software defect causing airbags to deploy on their own and seatbelts to tighten suddenly.
  - This stopped production for several days at Ford of Canada and other sites.
  - http://www.findarticles.com/p/articles/mi_m3165/is_2000_Oct/ai_68324491

Automotive Software Woes – 2

Source: http://www.leshatton.org/Documents/Hatton_Xmas1_03.pdf

- **20/May/2002.** 2000 top of the range BMW cars had to be recalled because of a software defect in the fuel injection pump.
  - http://www.heise.de/newsticker/data/uvo-28.05.02-002 (in German)

- **In September 2003**, Mercedes reported that they were reviewing early time to market in the wake of defects in automotive software systems which 'were proving hard to debug'.
  - http://www.autonewseurope.com/ (but unable to find this article)

- **May 2004;** Mercedes has $30 million recall of 680,000 cars due to defects in brake-assist-by-wire system. Blamed on hydraulics but the fix is applying a software patch.
  - http://www.automobil.md/news/comments.php?id=7

Automotive Software Woes – 3

- **March 2004: Chrysler Pacifica (34,561 vehicles)**
  - Software protocol used to test the vehicle exhaust gas recirculation (EGR) system may lead to engine stalling under certain circumstances, increasing the risk of a crash.
  - http://www.car-accident-advice.com/chrysler-pacifica-recalls-030004.html
- **Apr 2004: Jaguar recalls 67,798 cars for transmission fix**
  - Software defect slams car into reverse gear if there is a major oil pressure drop
  - http://www.accidentreconstruction.com/news/apr04/042104a.asp
- **Apr 2004: GM recalls 12,329 Cadillac SRXs**
  - One-second delay in brake activation "The problem, due to a software anomaly, only occurs during the first few seconds of driving when the SUV is moving slowly"
  - http://www.accidentreconstruction.com/news/apr04/040204a.asp
- **Dec 2004: Hyundai recalls 120,000 Elantras**
  - Airbag software problem detected in Insurance Institute crash tests (driver side airbag didn't deploy in crash test)
  - http://money.cnn.com/2004/12/19/pf/autos/bc.autos.hyundai.recall.reut/index.htm

Automotive Software Woes – 4

- **May 2005 Toyota recalls 23,900 Prius cars**
  - Hybrid car, engine dying in the middle of the highway
  - Requires a software upgrade due to a “programming glitch”
  - http://money.cnn.com/2005/05/16/Autos/prius_computer/index.htm

- **Feb 2010: CNN Headline: “Toyota: Software to blame for Prius brake problems”**
  - “Toyota officials described the problem as a "disconnect" in the vehicle's complex anti-lock brake system (ABS) that causes less than a one-second lag. With the delay, a vehicle going 60 mph will have traveled nearly another 90 feet before the brakes begin to take hold.”
  - “Brakes in hybrids such as the Prius operate differently from brakes in most cars. In addition to standard brakes, which use friction from pads pressed against drums or rotors, the electric motors in hybrids help slow them. The process also generates electricity to recharge the batteries.”
  - “The complaints received via our dealers center around when drivers are on a bumpy road or frozen surface, said Paul Nolasco, a Toyota Motor Corp. spokesman in Japan. "The driver steps on the brake, and they do not get as full of a braking feel as expected."
  - http://www.cnn.com/2010/WORLD/asiapcf/02/04/japan.prius.complaints/index.html?hpt=T1

Automotive Software Woes – 5

- **March 2010: Toyota vehicles cause Congressional inquiries**
  - Newer vehicles are throttle-by-wire
  - Concerns about runaway vehicles ➔ (See my Toyota UA talk from 2014)
    - http://embeddedgurus.com/barr-code/2011/03/unintended-acceleration-and-other-embedded-software-bugs/

- **April 2010: Toyota recalls Lexus GX 460 SUVs**
  - Consumer reports rated "do not buy" due to rollover risk uncovered during testing
  - Toyota recalled 9,400 in US; 34,000 worldwide, and suspended sales.
  - Vehicle Stability Control software fix
    - http://money.cnn.com/2010/04/19/autos/lexus_gx460_recall/index.htm

- **Nov 2011: Ford sends 250,000 flash drives with software upgrades to MyFord Touch**
  - Problems are said to be responsible for dramatic downtrend in quality perception
  - Not sure how to upgrade 200K buyers outside US
  - "the company also learned quickly that buyers aren't as forgiving with glitches in their cars as they are with their phones or computers." [Durbin AP, Manufacturing.Net Nov 7, 2011]
    - http://www.manufacturing.net/News/2011/11/Electrical-Electronics-Ford-To-Upgrade-MyFord-Touch-After-Taking-Heat/?

Jaguar recalls 18,000 cars over cruise control software fault

Car system upgrade needed, but no hardware affected

By Leo King | Computerworld UK | Published 10:23, 24 October 11

Jaguar has recalled nearly 18,000 X-type cars after it discovered a major software fault, which meant drivers might not be able to turn off cruise control.

The problem lies with engine management control software developed in-house by Jaguar. The problematic software is only installed on diesel engine X-Types, which were all produced between 2006 and 2010.

Some 17,678 vehicles have been recalled, as a result of the potentially dangerous problem. If the fault occurs, cruise control can only be disabled by turning of the ignition while driving - which would mean a loss of some control and in many cars also disables power steering. Braking or pressing the cancel button will not work.

"Jaguar has identified that should an error with certain interfacing systems be detected the cruise control system will be disabled and an error message displayed to the driver on the instrument cluster," the company said in a statement.

Jaguar said drivers who returned their cars would need a software upgrade to their vehicle. No hardware needed to be replaced, it said.

The company wrote to all owners of X-types, warning them that they might not be able to switch off cruise control when they are driving. However, it insisted no customers have reported the problem, adding that one employee identified the issue on his own car.

'No customer has been affected and there had been no accidents or injuries," Jaguar said. The company did not immediately provide further detail on the software causing the problem. *Photo credit: Jaguar*

Automotive Software Woes – 6

- **Honda recalls nearly 350k Odyssey minivans over unintended braking**
  - The issue revolves around a combination of parts and software that have been reported to cause the vehicle to brake hard and unexpectedly, without illuminating the brake lights.
    - http://www.autoblog.com/2013/11/03/honda-odyssey-recall-unintended-braking/

- **Jan 2014: General Motors recalls 370,000 GM, Chevy pickups with engine fire risk**
  - The trucks are only supposed to use two cylinders when idling, but a software glitch is causing them to idle with most of their cylinders. This can cause exhaust components to overheat, and hence potentially catch fire.
  - 3 fires on customer-owned vehicles
    - http://www.csmonitor.com/Business/In-Gear/2014/0113/General-Motors-recalls-370-000-GM-Chevy-pickups-with-engine-fire-risk

- **Nov 2015: Tesla software can cause Mercedes B-Class Electric Drive stalling**
    - http://ecomento.com/2015/11/04/tesla-software-can-cause-mercedes-b-class-electric-drive-stalling/

Automotive Software Woes – 7

- **Feb 2014: Toyota recalls 1.9M Prius cars due to software bug**
  - Hybrid vehicles with plenty of software controlling vehicle
  - SW overheats/damages power electronics; vehicle shuts down while driving
    - http://bostonherald.com/business/automotive/2014/02/toyota_recalls_19m_prius_cars_for_software_glitch
  - Also 294,000 vehicles for skid control software problem
    - http://www.washingtonpost.com/business/technology/toyota-recalls-vehicles-in-us-for-software-glitch/2014/02/13/a7cff5ba-9481-11e3-9e13-770265cf4962_story.html

- **March 2014: Nissan Recalls 1M vehicles due to airbag flaw**
  - 2013-2014 Nissan & Infiniti models
  - Shuts off airbag via weight sensor SW bug even if adult is in the seat
    - http://money.cnn.com/2014/03/26/autos/nissan-recall/

Toyota Case: Single Bit Flip That Killed

Junko Yoshida

10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including bugs that can cause unintended acceleration.

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they "uncovered gaps and defects in the throttle fail safes."

The experts demonstrated that "the defects we found were linked to unintended acceleration through vehicle testing," Barr said. "We also obtained and reviewed the source code for the black box and found that it can record false information about the driver's actions in the final seconds before a crash."

Stack overflow and software bugs led to memory corruption, he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of tasks' deaths studied by the experts in their experiments "were not detected by any fail safe."

Bookout Trial Reporting

http://www.eetimes.com/document.asp?doc_id=1319903&page_number=1 (excerpts)

"Task X death in combination with other task deaths"

Honda Admits Software Problem, Recalls 175,000 Hybrids

Junko Yoshida

7/10/2014 03:05 PM EDT

MADISON, Wis. — For the first time, an automobile company has conceded that a software glitch in electronic control units could cause cars to accelerate suddenly, forcing drivers to scramble to take emergency measures to prevent an accident. Honda Motor Co., citing software problems, announced Thursday that it is recalling 175,000 hybrid vehicles in Japan.

Honda revealed that some hybrid versions of its Fit and Vezel subcompacts could suddenly accelerate without warning.

Software Isn't The Only Thing That Breaks

- **May 2000: Firestone tire recall is perhaps the most deadly auto safety crisis in American history**
  - (Software isn't the only thing that breaks)
  - Estimated 250 deaths from rollover after tire failure
    - Treads peel off casings, problems with hot weather & high speed
    - Tires standard on Ford Explorer
  - Legislation which mandates the installation of safety equipment. (TREAD) Act of 2000
    - Tire pressure sensors
    - http://www.firestone-tire-recall.com/pages/overview.html

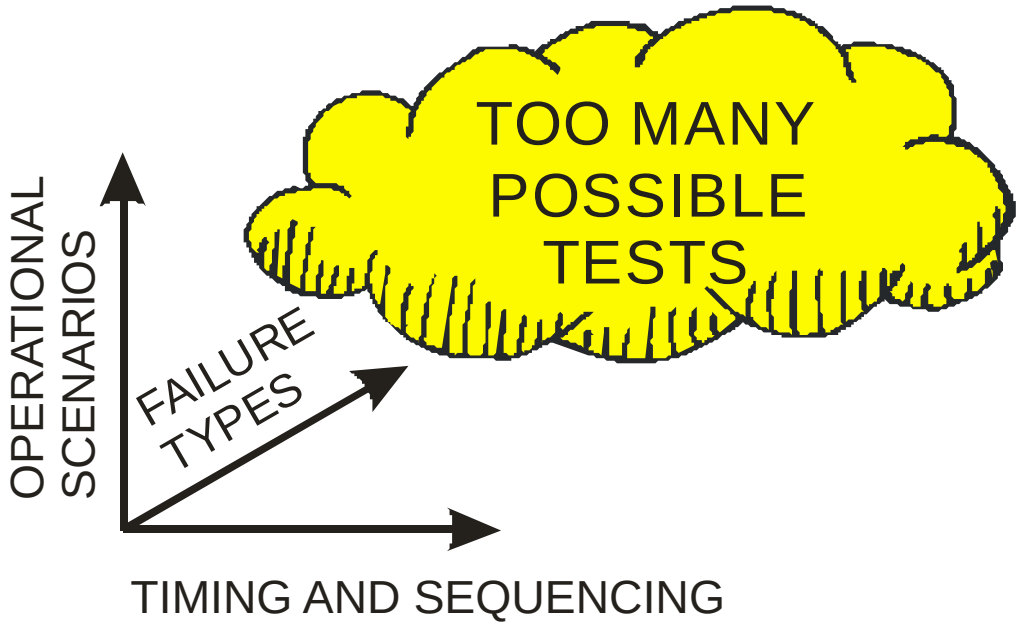
Overview

- **Candidate automotive embedded safety critical development processes:**
  - MISRA
    - Automotive specific; but 20 years old
  - IEC 61508
    - More generic, broad acceptance in non-automotive industry
    - But, new variant would be desirable for automotive (especially X-by-Wire)
  - ISO 26262 – newer standard
    - Which is a significant adaptation of IEC 61508 to for automotive applications

- **You need a defensible process for creating safe software**
  - Consider adopting documented best practices instead of inventing your own
    - If you adopt your own, be prepared to demonstrate it is as good as standards
  - If everyone else adopts MISRA or IEC 61508 and you don't, you might be considered negligent (failure to follow "standard practices")

Why Can't We Just Test Until Car Is Safe?

- **Vehicle level testing is useful and important**
  - Can find unexpected component interactions
- **But, it is impracticable to test everything at the vehicle level**
  - There are too many possible operating conditions
  - There are too many possible timing sequences of events
  - There are too many possible faults
  - All possible combinations of component failures and memory corruptions
  - Multiple software defects activated by a sequence of operations

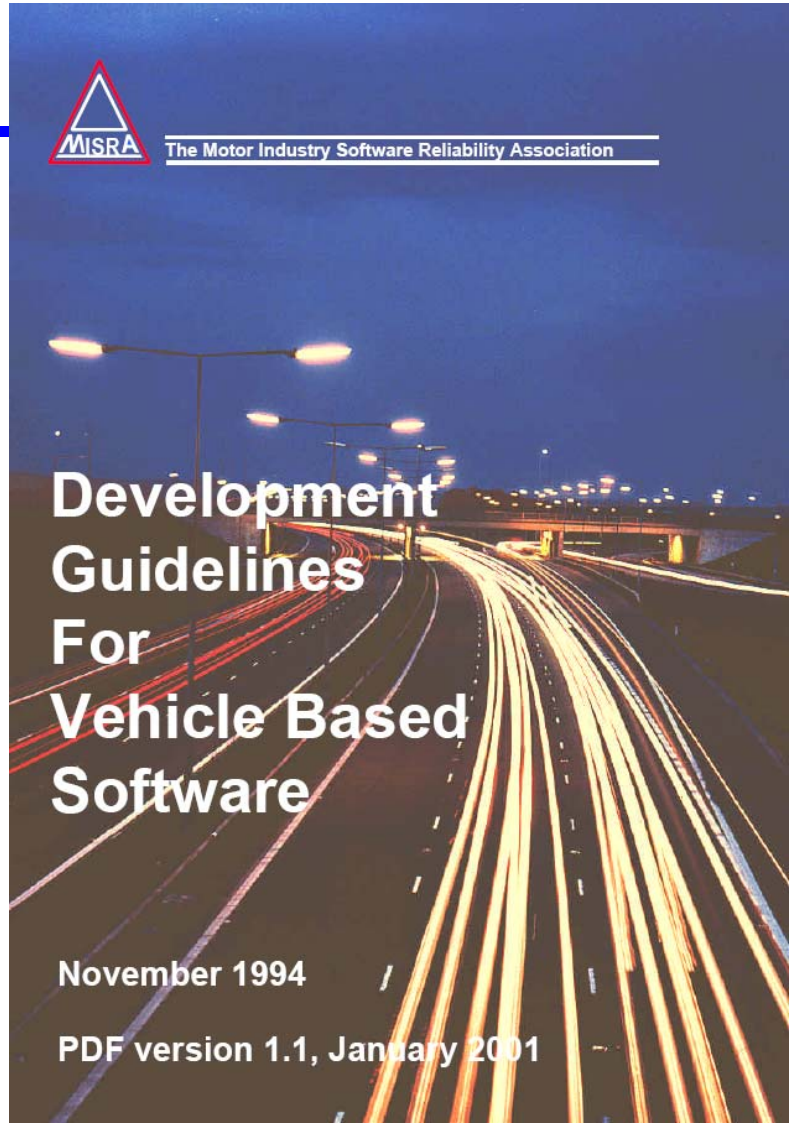


MISRA

- **The Motor Industry Software Reliability Association**
  - Includes largely UK automotive organizations
  - Has published recommended practices for safe automotive software
  - Overall guidelines and detailed reports

- **You can register for & upload reports**
  - ***http://www.misra.org.uk/license.htm***

  Click “accept license” button at bottom to continue
  - Report 2 on Software Integrity is the most important report for this lecture



MISRA Documents Overview

http://www.misra.org.uk/ download reports for free with registration

- **Overview Document:**
  "Development Guidelines for Vehicle Based Software", November 1994
  - Due for update “in 2004” – looks like they are moving to adopt IEC 61508

- **Report with details on various areas:**
  - MISRA Report 1, "Diagnostics and Integrated Vehicle Systems", February 1995.
  - MISRA Report 2, "Integrity", February 1995.
  - MISRA Report 3, "Noise, EMC and Real-Time", February 1995.
  - MISRA Report 4, "Software in Control Systems", February 1995.
  - MISRA Report 5, "Software Metrics", February 1995.
  - MISRA Report 6, "Verification and Validation", February 1995
  - MISRA Report 7, "Subcontracting of Automotive Software", February 1995.
  - MISRA Report 8, "Human Factors in Software Development", February 1995.
  - MISRA Survey Report, "Sources of Information", February 1995.

  - *Guidelines for the Use of the C Language in Vehicle Based Software* - also known as "MISRA C" - (published April 1998)
    - This one costs money; it is how they make enough money to keep everything else running

MISRA Basic Safety Principles – 1

(Commentary bullets are mine; not part of MISRA documents)

a) Safety must be seen to be present (i.e., safety cases must be public)

- Experience shows that public scrutiny of safety cases is required
- A proprietary safety case must be presumed to be hiding defects
- "Public" might just be access to independent certification authorities, although truly public is even better

b) The greater the risk, the greater the need for information

- More Verification & Validation is required for greater risks

c) Software robustness, reliability and safety, like quality, should be built in rather than added on

- High quality usually cannot be achieved via just fixing defects found by testing
- Specific and significant design effort must be spent on dependable/high integrity from the very first day of any software project

d) The requirements for human safety and security of property can be in conflict. Safety must take precedence

- Better to have a car totaled than to risk loss of life
- Shows up in approaches such as crumple zones (more body damage/less people damage)

MISRA Basic Safety Principles – 2

(Commentary is not part of MISRA documents)

e) System design should consider both random and systematic faults

- Random hardware faults are to be expected
- Design defects (especially software defects) are to be expected in an operating vehicle; so design for safety even if defects occur

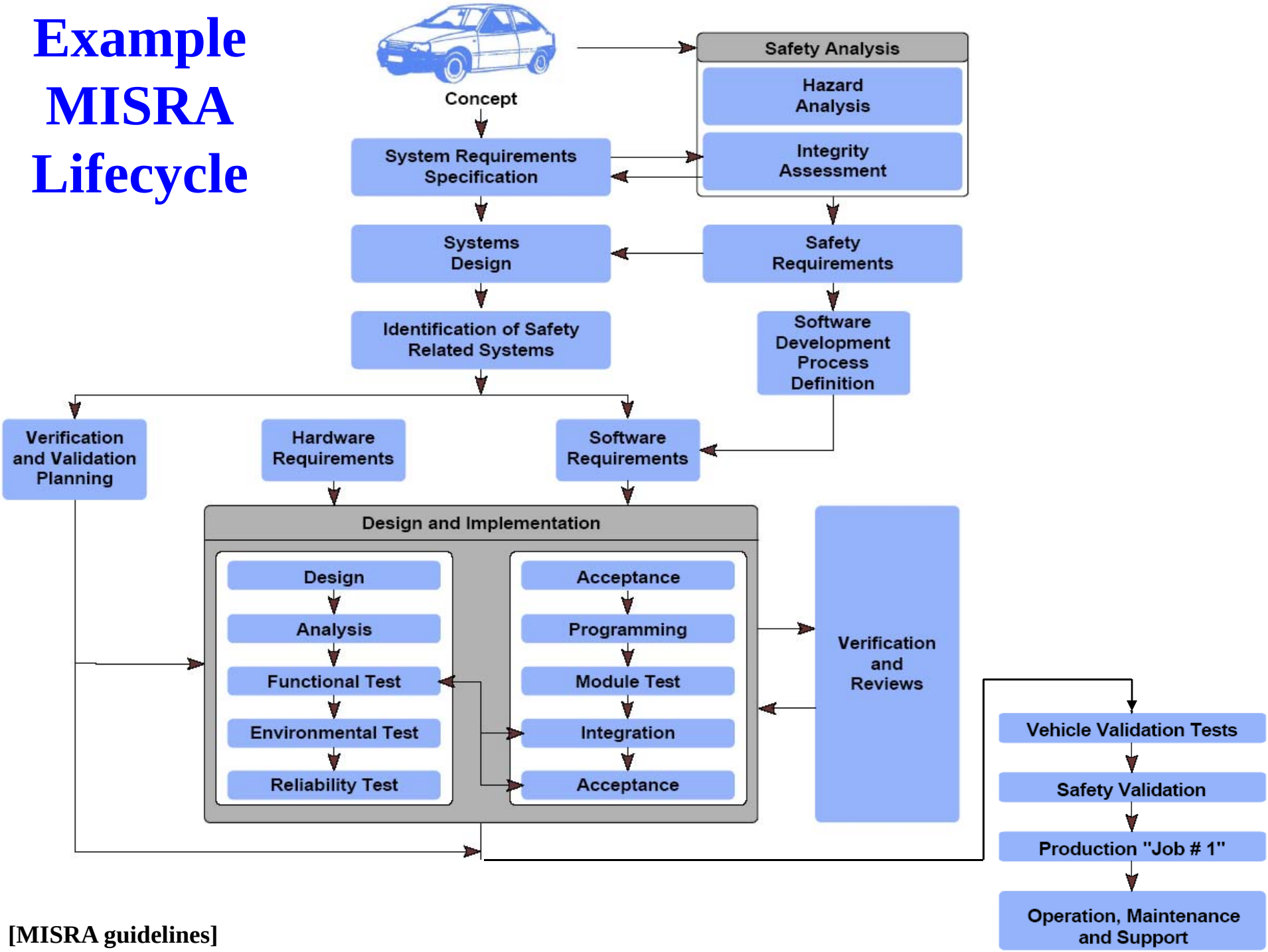
f) It is necessary to demonstrate robustness, not rely on the absence of failures

- Simply observing no problems during functional testing is insufficient
- Verification & Validation must actively evaluate fault responses (e.g. fault injection)

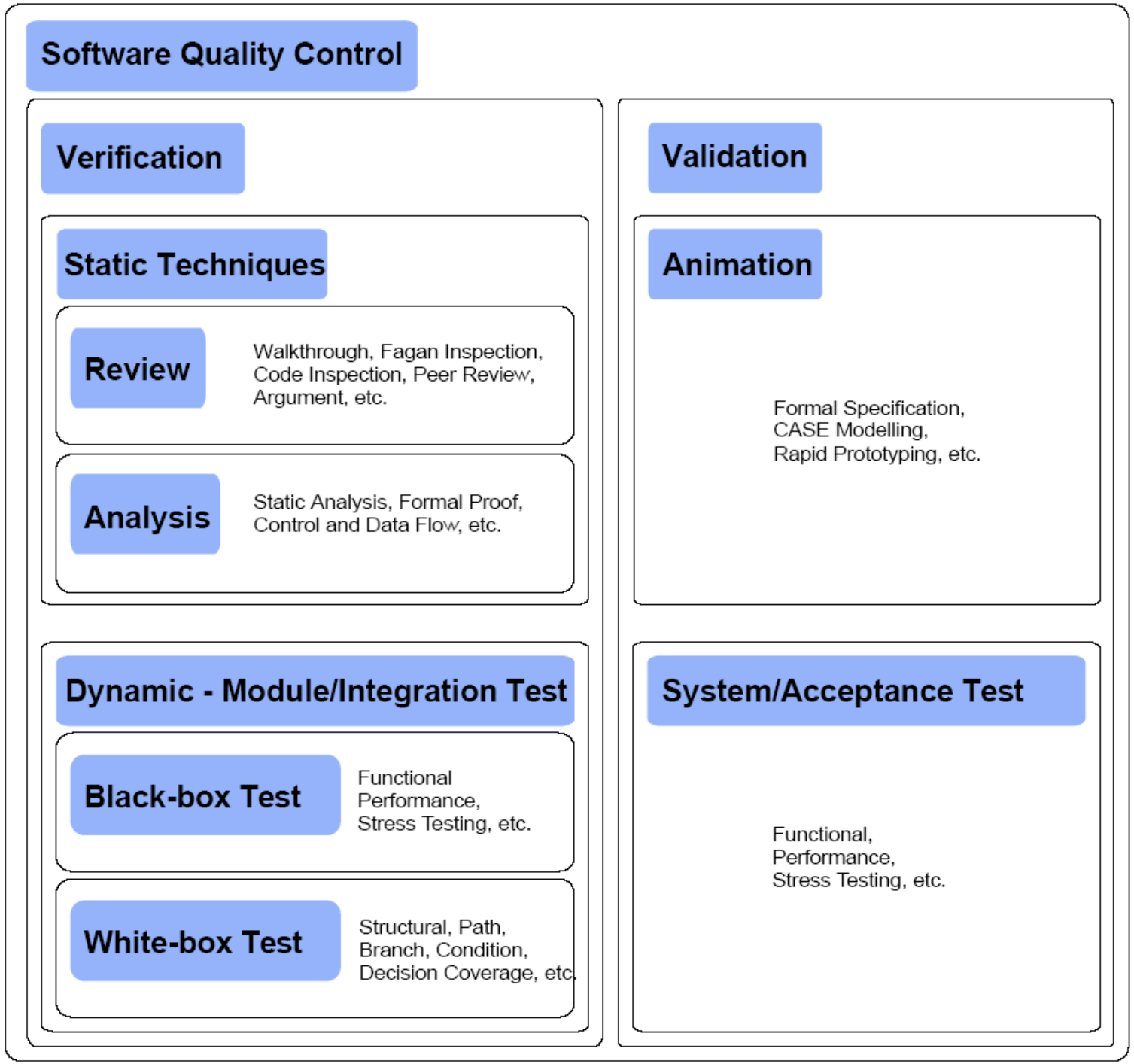
g) Safety considerations should apply across the design, manufacture, operation, servicing and disposal of products

- Concern for safety doesn't end when the product is shipped

Example MISRA Lifecycle



[MISRA guidelines]



[MISRA guidelines]

Software Integrity

- **Integrity is a notion similar to dependability + safety**

- **Integrity needed to avoid:**
  - Harm to humans
  - Breaking legislated laws & regulations
  - Undue traffic disruption
  - Damage to property (e.g., collisions)
  - Damage to environment (e.g., emissions)
  - Undue financial loss to manufacturer or owner

- **Level of integrity should vary corresponding to risk**
  - High levels of integrity required for high risk
  - Lower levels of integrity acceptable for low risks
  - Notes:
    - RISK is more or less the traditional expected value of probabilities * losses
    - Low integrity doesn't mean "definitely bad" – it means we can't prove it is good

Safety Integrity Level (SIL)

- **Five levels that determine how critical system or component is**
  - Ranked according to possibility of safe recovery from a fault
  - Similar approach to aviation & rail safety standards

SIL	Controllability	Acceptable Failure Rate
4	Uncontrollable	Extremely Improbable
3	Difficult To Control	Very Remote
2	Debilitating	Remote
1	Distracting	Unlikely
0	Nuisance only	Reasonably possible

- **If multiple potential faults have different levels of controllability:**
  - Highest single controllability risk determines SIL
  - Can use analysis (e.g., FTA) to limit high SIL requirements to a few components

What Do The Controllability Ratings Mean?

These are informal summaries – see MISRA guidelines for details

Examples are also informal; interpretations might vary

- **Uncontrollable – SIL 4** "critical failure"
  - No driver can be expected to recover from this fault
    - E.g., loss of normal braking, parking brake, and steering simultaneously
  - Usually extremely severe outcomes, such as multi-vehicle fatal crash
- **Difficult to control – SIL 3** "critical failure"
  - Good driver can recover in a favorable situation
    - E.g., loss of normal braking; but parking brake still works
  - Usually very severe outcomes, such as fatal crash
- **Debilitating – SIL 2**
  - Ordinary driver can recover most of the time
  - Reduction in safety margin, but usually no worse than severe outcome
- **Distracting – SIL 1**
  - Operational limitations, but minor problem for any licensed driver
- **Nuisance only – SIL 0**
  - Safety is not an issue; fault is only an issue of customer satisfaction

How Often Is Improbable?

- **MISRA Report 2 Appendix A gives a complex procedure**
  - There is no simple way to summarize the numbers
  - But, the outcome is likely to be similar to aviation in the end, so use those as a starting point to set expectations.
  - **CAUTION!** – these are not the MISRA numbers; they just give a feel for the situation

AVIATION failure rates with inexact analogy to automotive situations:

- **Catastrophic** (10^{-9}/hr) => "uncontrollable" ~ SIL 4
  - Prevents continued safe flight and landing
- **Hazardous** (10^{-7}/hr) => "difficult to control" ~ SIL 3
  - Large reduction in safety margins; perhaps fatal injuries to some passengers
- **Major** (10^{-5}/hr) => "debilitating" ~ SIL 2
  - Significant reduction in safety margins; perhaps non-fatal injuries to passengers
- **Minor** (10^{-3}/hr) => "distracting" ~ SIL 1
  - Slight reduction in safety margins; reasonable workarounds
- **Nuisance** (10^{-2}/hr) => "nuisance only" ~ SIL 0
  - Failure causes no effect

Real Vehicles Have Drivers

- **Driver abilities/situation must be factored into overall Risk**

- **Human operator issues apply, just as in aviation & nuclear power:**
  - Human reaction times
  - Ease of recognition of a situation
  - Attentiveness
  - Driver experience
  - Risk compensation (improved safety can lead to riskier behavior)
  - Subversion or overriding of system safety features/functions
  - Smooth and readily perceived transfer of control from system to driver
  - Workload of driver, especially at moment of failure

Approach To Avoid Hazardous Malfunctions

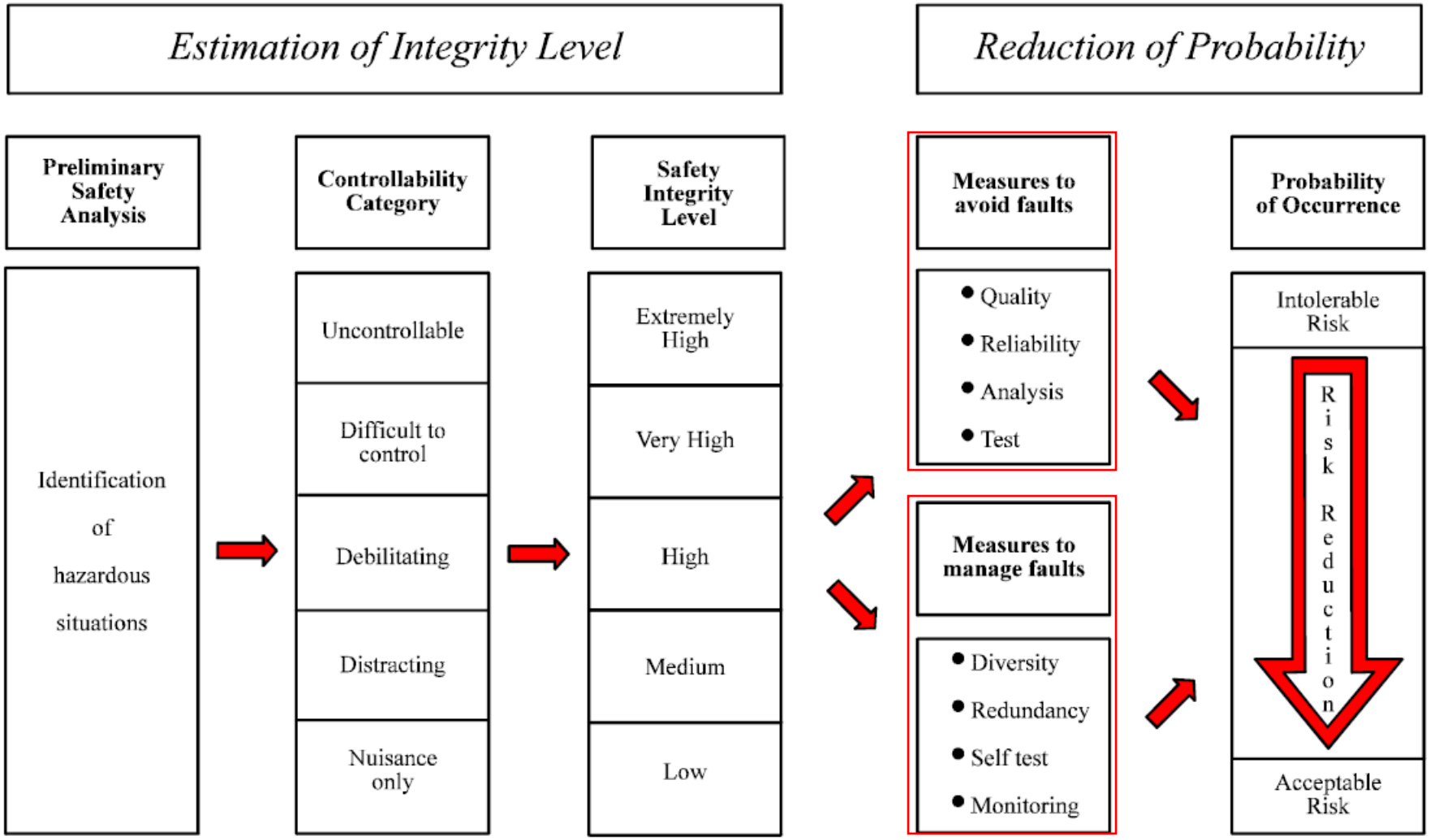


Figure 4 - Approach to avoid hazardous malfunctions

[MISRA Report 2]

How Do You Implement SW At A Given SIL?

- **For each SIL, there are specific requirements**
  - Specification & design
  - Languages & compilers
  - Configuration management products
  - Configuration management processes
  - Testing
  - Verification & validation
  - Access for assessment

- **In general, SILS 3 & 4 are critical**
  - SILS 1 & 2 require care, but 3 & 4 are treated as dramatically more critical
  - Every SIL requires lower SIL activities plus, usually, additional activities

- **Important notes:**
  - Technical requirements go beyond this list. For example, proof of safety properties may require many techniques we've discussed to actually accomplish
  - IEC 61508 has a more extensive list; it would be no surprise if a new version of MISRA incorporates some of those as well

Development Process	Integrity Level				
	0	1	2	3	4
Specification and design	ISO 9001	Structured method.	Structured method supported by CASE tool.	Formal specification for those functions at this level.	Formal specification of complete system. Automated code generation (when available).
Languages and compilers		Standardized structured language.	A restricted subset of a standardized structured language. Validated or tested compilers (if available).	As for 2.	Independently certified compilers with proven formal syntax and semantics (when available).
Configuration management: products		All software products. Source code.	Relationships between all software products. All tools.	As for 2.	As for 2.
Configuration management: processes		Unique identification. Product matches documentation. Access control. Authorized changes.	Control and audit changes. Confirmation process.	Automated change and build control. Automated confirmation process.	As for 3.

Development Process	Integrity Level				
	0	1	2	3	4
Testing	ISO 9001	Show fitness for purpose. Test all safety requirements. Repeatable test plan.	Black box testing.	White box module testing — defined coverage. Stress testing against deadlock. Syntactic static analysis.	100% white box module testing. 100% requirements testing. 100% integration testing. Semantic static analysis.
Verification and validation		Show tests: are suitable; have been performed; are acceptable; exercise safety features. Traceable correction.	Structured program review. Show no new faults after corrections.	Automated static analysis. Proof (argument) of safety properties. Analysis for lack of deadlock. Justify test coverage. Show tests have been suitable.	All tools to be formally validated (when available). Proof (argument) of code against specification. Proof (argument) for lack of deadlock. Show object code reflects source code.
Access for assessment		Requirements and acceptance criteria. QA and product plans. Training policy. System test results.	Design documents. Software test results. Training structure.	Techniques, processes, tools. Witness testing. Adequate training. Code.	Full access to all stages and processes.

Need Adequate Technical Approaches Too

- **Activities to achieve safety with a SIL approach:**
  - Determine SIL based on severity of a failure
  - Follow SIL-appropriate development practices
  - Follow SIL-appropriate technical practices
  - Follow SIL-appropriate validation practices
  - Make sure process is really working

BASED ON SIL:
☑ DEVELOPMENT
☑ TECHNICAL
☑ VALIDATION
☑ PROCESS QUALITY

- **Follow accepted practices elsewhere:**
  - No single point of failure (see later in lecture)
  - Real time scheduling that ensures deadlines met
  - Watchdog timer that really works
  - Code that can be tested and reviewed (good modularity; acceptable complexity)
  - SQA and good practices throughout lifecycle
  - Good safety culture (do you take safety seriously? Or just go through the motions?)

Important Notes On Following Slides

- **High level bullets are quotes from previous MISRA tables**

- **Other material is *opinion*, not hard-and-fast rules**
  - In some particular cases, requirements might be relaxed
  - In some cases, requirements might be more stringent
  - In all cases, designers are responsible for creating safe systems regardless of MISRA rules/guidelines/opinions

- **Important high-level *potential* conclusions to consider**
  - In general, everyday software development tools have to really stretch to reach SIL 3
    - Requires skills in formal methods
    - Requires specialists in all areas, including testing & safety cases
  - Probably you can't write SIL 4 software without multiple Ph.D.s on staff who specialize in safety critical software

Specification & Design

- **SIL 1: Structured method**
  - UML is fine
  - Generally, any methodical design approach is probably OK
  - Pencil & paper is OK

- **SIL 2: Structured method supported by CASE tool**
  - Some tool is used to help with structured method
  - Simple examples: Statemate, Simulink
  - More complex example: code synthesis from UML tool

- **SIL 3: Formal specification of those functions at this level**
  - For example, specification of SIL 3 functions in Z
  - Some areas can be difficult, such as expressing real time behavior

- **SIL 4: Formal specification of complete system. Automated code generation (when available).**
  - This is a research topic. Apparently some applications do this.
  - Job is made easier by encapsulating SIL 4 behavior in smallest system possible

Languages & Compilers

- **SIL 1: Standardized structured language**
  - Use a language without “goto” statement, e.g., C rather than BASIC
  - Assembly language is, in general, not structured!
  - That means, in general, assembly language is outlawed even for SIL 1

- **SIL 2: A restricted subset of a standardized structured language. Validated or tested compilers (if available).**
  - “Validated” compiler (really = “certified”) means high assurance that compiled code represents source code (“no” compiler defects)
  - MISRA C (more on this later; it isn’t really strongly typed)
  - Preference for strongly typed languages
  - Ada (especially Spark Ada, a subset of Ada)
    - http://www.sparkada.com
    - All Ada compilers are validated
- **SIL 3: as for SIL 2**

Languages & Compilers – 2

- **SIL 4: Independently certified compilers with proven formal syntax and semantics (when available)**
  - In general, C and C subsets are not strong on proven formal syntax and semantics
  - Issue is that different compilers can interpret code in different ways (ambiguity in language specification)
  - http://www.praxis-cs.co.uk/sparkada/pdfs/misracatsil4reader.pdf talks about MISRA SIL 4 in particular
  - In absence of formally proven compiler, might need to show that output corresponds to input
    - This implies re-verifying output code EVERY TIME IT IS COMPILED!

An Aside – MISRA C

- **Guidelines that loosely define a subset of C programming language**
  - $76 document; 50 pages; *NOT* available on web for free
  - Updated in 2004; 2012

- **"Common sense" rules that belong in most coding style sheets**
  - **Rule 20 (required): All object and function identifiers shall be declared before use.**

- **Some stylistic rules that are a matter of opinion, but must be followed**
  - **Rule 49 (advisory): Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.**

- **Rules that are designed to reduce number of "stupid" errors**
  - **Rule 104 (required): Non-constant pointers to functions shall not be used.**

- **Off-the-shelf tools have MISRA C checkers (e.g., PC-LINT)**
  - You should at least use MISRA C for safety critical software

MISRA C Evolves with Time (so should your rules)

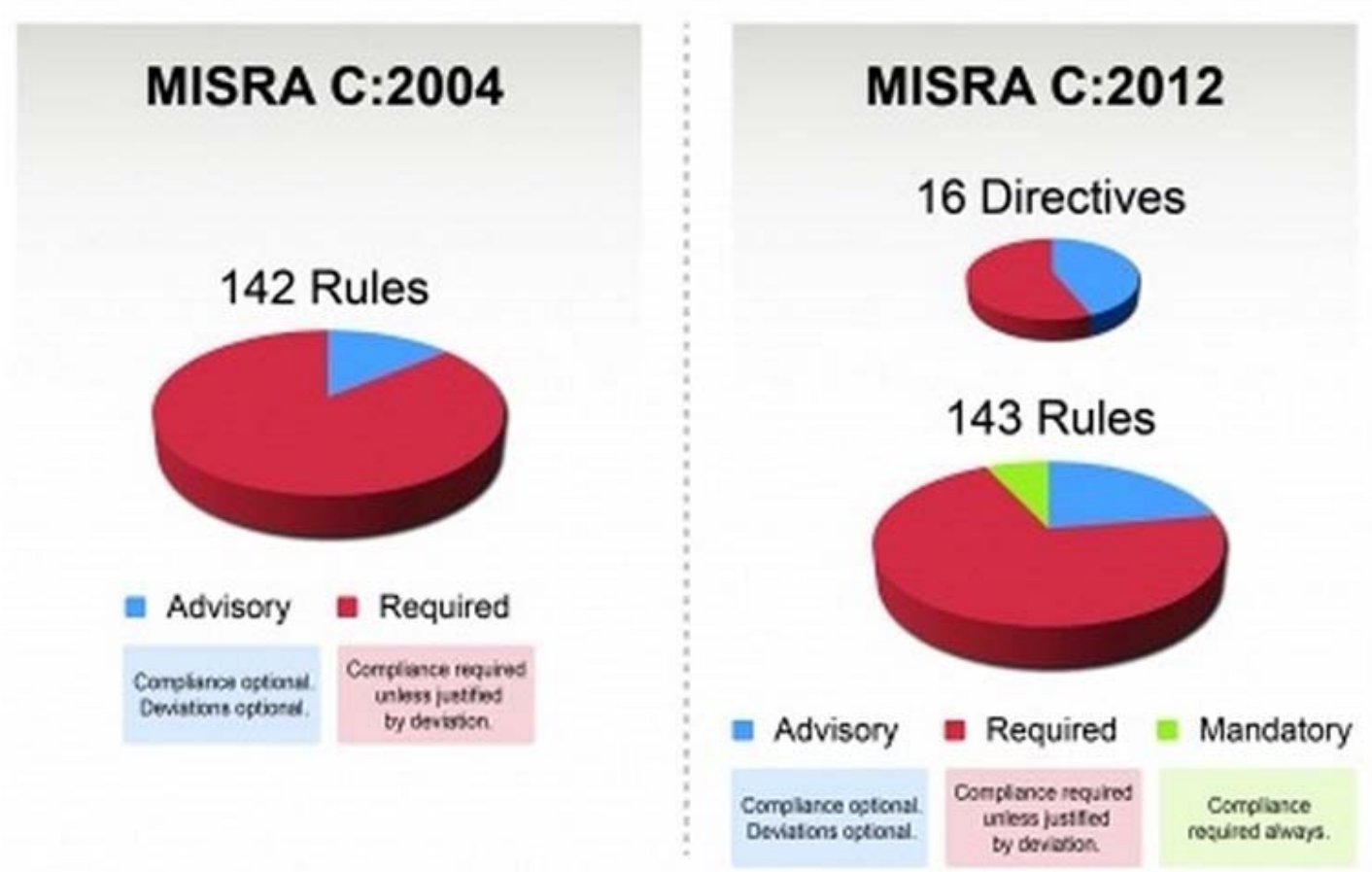


Image: Old vs. New: MISRA C:2012 incorporates a more differentiated approach towards software engineering

[Burden13]

Configuration Management: Products

- **Configuration management areas:**
  - Uniquely identify versions of each software item
  - Identify versions of each item that constitute a complete product build
  - Identify build information for all products in development; delivered; installed
  - Control simultaneous updating of software by multiple developers
  - Provide coordination for updating multiple products at multiple locations
    - This includes subcontractors!
  - Identify and track all actions and changes from change requests – initiation through release

- **SIL 1: All software products. Source code**
  - Source code, designs, documents, manuals

- **SIL 2: Relationships between all software products. All tools.**
  - All dependencies (e.g., module A version 6 requires module B version 3.2)
  - All tools under configuration management (necessary to rebuild old versions)
    - As a practical matter, includes old OS versions; maybe even old hardware

- **SILs 3 & 4: As for SIL 2**

Configuration Management: Processes

- **SIL 1: Unique identification. Product matches documentation. Access control. Authorized changes.**
  - Previous list of configuration management areas implemented
  - Ensure product matches documentation (e.g., design matches implementation)
  - Access control to ensure unauthorized people don't change pieces of software
  - Changes approved by configuration control board, not just at programmer's whim

- **SIL 2: Control and audit changes. Confirmation process.**
  - Audits of control and change processes by independent auditors
  - Confirmation means ensuring that installed software actually matches what configuration management says it should be

Configuration Management: Processes – 2

- **SIL 3: Automated change and build control. Automated confirmation process.**
  - Changes automatically logged; automated version control system
  - Software builds are done in a completely automated manner
    - Makes it impossible to forget to do a build step for release software
  - Confirmation that installed software is correct configuration is automatic

- **SIL 4: As for SIL 3**

Testing

- **SIL 1: Show fitness for purpose. Test all safety requirements. Repeatable test plan.**
  - Test plan must be written and give repeatable results. Can be manual testing
    - Tests should be traceable to something (not necessarily requirements)
  - Acceptance test must provide reasonable coverage of normal operating scenarios (concept of operational profile applies)
  - All safety requirements must be traceable to some test (preferably acceptance test)

- **SIL 2: Black box testing.**
  - Functional (behavioral)/black box testing performed of system

Testing – 2

- **SIL 3: White box module testing – defined coverage. Stress testing against deadlock. Syntactic static analysis**
  - Defined, defensible structural (white box) testing coverage. (95%+ is realistic)
  - Multi-tasking stress tests to (hopefully) expose deadlocks and other concurrency issues
  - Syntactic static analysis is examination of code for structural problems
    - Lint or high quality compiler warning analysis (maybe with multiple compilers)
    - Can also include coding style design review

- **SIL 4: 100% white box module testing. 100% requirements testing. 100% integration testing. Semantic static analysis.**
  - "100%" white box testing (probably this means branch coverage)
  - 100% of requirements are traceable to tests
  - 100% data connectivity integration testing (every data element transmitted to every module)
  - Semantic static analysis
    - Includes strong type checking, and possibly other tools such as array bound checks

Verification & Validation

- **SIL 1: Show tests: are suitable; have been performed; are acceptable; exercise safety features. Traceable correction.**
  - Tests actually test the things that matter. At a minimum, backward traceability:
    - White box tests backward traced to design & implementation
    - Black box tests backward traced to design & requirements
  - Test logs indicate test plan performed, including results of tests
  - Test results are "pass" OR results corrected
    - If "too many" acceptance tests fail at higher SILs, this raises question of software quality
  - 100% of safety features trace to tests
    - Means every safety feature tested at least one way
    - Does not mean safety has been completely tested!

- **SIL 2: Structured program review. Show no new faults after corrections.**
  - Design reviews of, at a minimum, the actual code
  - Some form of regression testing to catch fault reinjection effects

Verification & Validation – 2

- **SIL 3: Automated static analysis. Proof (argument) of safety properties. Analysis for lack of deadlock. Justify test coverage. Show tests have been suitable.**
  - Static analysis of formally specified code (e.g., type checkers and more)
  - Safety case
  - Analysis for lack of deadlock livelock (testing alone isn't good enough)
  - If any test metric is less than 100%, justify why system is safe
    - (From testing lecture, recall that 100% coverage on every metric is impossible)
  - Ensure that all interactions of safety features have been covered
  - Show that tests confirm the validity of formal analysis of formal specifications
    - IMPORTANT – this is a shift from testing as finding defects to testing as a quality check on creating "perfect" software!

Verification & Validation – 3

- **SIL 4: All tools to be formally validated (when available). Proof (argument) of code against specification. Proof (argument) for lack of deadlock. Show object code reflects source code.**
  - Software tools formally validated (if such tools exist)
    - This ensures that what designers put into tools matches generated code
    - Don’t forget that this applies to any hardware synthesis tools as well!
  - Formal proof/safety case of formally specified code against its specification
    - In other words, need to prove that generated code really meets specification
  - Prove that object code matches source code (even if compiler is validated)
  - Proof/safety case against livelock and deadlock

Access For Assessment

If a third party can't assess safety,
then your system can't be considered safe.

- OEMs will, probably, increasingly require 3rd party assessments

- **SIL 1: Requirements and acceptance criteria. QA and product plans. Training policy. System test results.**
  - Copies of written requirements and acceptance tests/criteria
  - QA plan, product development plan, system test results
  - Training policy (how do you know developers have appropriate safety skills?)

- **SIL 2: Design documents. Software test results. Training structure.**
  - Complete end-to-end design documents (including available traceability analysis)
  - Software test results (and, probably, all V&V paperwork)
  - Inspection of company training (make sure training policy is being carried out)

Access For Assessment – 2

- **SIL 3: Techniques, processes, tools. Witness testing. Adequate training. Code.**
  - Written documentation of techniques, all software processes, and standard tools
  - Ability to audit conformance to techniques, processes, use of tools
  - Testing of system in presence of a witness (e.g., 3rd party witness to acceptance testing)
  - Audit/assess whether developers really possess skills they have been trained to have
  - Complete access to all source code

- **SIL 4: Full access to all stages and processes.**
  - At a minimum, ability to ask any question and see any document
  - Preferred – absolutely everything necessary to perform audit is in written form

Other Items Not In Table

- **The table doesn't say much about fault tolerance**
  - It is really buried in the "safety case" bullets and Report 2 Appendix E
  - In general, need extensive self-checks and/or redundancy
  - (excerpt from Rpt. 2 App. E):

- Watchdog
- Comparator
- Voter
- Other Units

- Parity Bit Technique
- Multi-Bit Redundancy Technique
- Longitudinal Redundancy Check (LRC), Simple Check Word
- Cyclic Redundancy Check (CRC), Simple Check Word
- Cyclic Redundancy Check (CRC), Multiple Check Word

Software

- Plausibility Check
- Time Monitoring of Program Tasks
- Watchdog
- Logical Monitoring of Program Sequences
- Combination of Time and Logical Monitoring of Program Sequences
- Diversity of Software

"Reality Check" Time

- **Consider a SIL 4 system:**
  - Should be huge MTBF for critical software defects (something like 10^9 hrs)

- **What is the safety argument being made?**
  - Determine that SIL 4 is applicable
  - Use listed techniques

  - BUT, there is no measurement that the result actually achieves 10^{-9}/hr !
  - Rather, the approach is designed keeping in mind affordability vs. risk
    - What it is really saying is, do everything reasonable for the integrity level
  - AND, it assumes that you are trying as hard as possible to create safe code given the processes you are using
  - It is POSSIBLE that these lists aren't really good enough
    - The argument is that they are all automotive can afford; not that they are sufficient
    - But, also, that the industry overall thinks that these are good enough, as far as we know

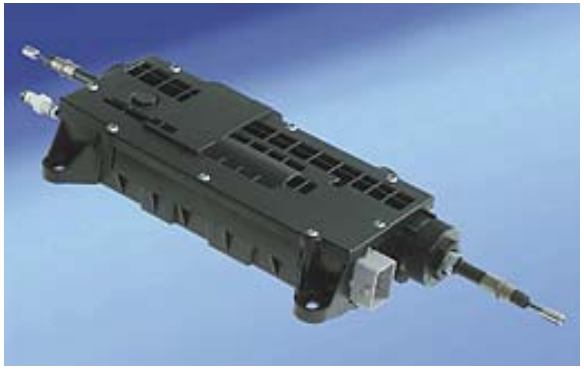
- **What is a guess for the future?**
  - Almost certainly SIL 4 will get harder every time there is a major problem
    - The goal is, for practical purposes, zero defects for critical software
  - SIL 3 will probably adopt previous SIL 4 techniques as they become common
  - Research community moving to "safety arguments" (see later slide)

Example Discussion: Electronic Parking Brake

http://www.conti-online.com/generator/www/de/en/continentalteves/continentalteves/themes/products/electronic_brake_systems/parking_brake_1003_en.html

Possible EPB Functions:

- Emergency braking if primary brakes fail
- "Drive-away" automatic release on hills
- Normal parking brake function
- Vehicle immobilizer (car security system)



Discussion questions:

Assume critical functionality is provided by software

- What are the worst potential hazards?
- What is the SIL of this system?
- What is a likely acceptable failure rate?
- What architecture might be acceptable to provide this failure rate?
- Who is responsible for ensuring safe operation within design flow?

Overall Themes

- **In some cases the tools/techniques for SIL 4 are problematic**
  - MISRA guidelines say “if available”
  - This means you have to have a substitute if tools are unavailable
    - Example: manual verification of output code if compiler isn’t formally proven
  - It also means if something becomes available, you’ll have to adopt it
    - Failure to keep up with best practice/available tools could be a legal liability!

- **Simply following the rules blindly isn’t good enough**
  - Time and again, experience shows that software safety requires “safety culture” in the industry, supplier, and OEM
    - The reality is there is very little SIL 4 automotive software deployed
    - If you don’t have to use “unavailable” tools, then are you really as safe as you are supposed to be?
    - Really, what SIL 4 is saying is use everything you can afford
  - When SIL 4 software starts being common, the rules are likely to change
    - They are likely to change by becoming stricter!

SIL-Like Approach Is Common

All have a catalog of techniques for risk reduction

- **Rail – European (and US) train systems CENELEC ENV-50128/50129**
  - "Vital" and "non-vital" processes, with SILS 1-4
  - Tables of "recommended" and "highly recommended" practices, just like MISRA
- **FDA – US medical software**
  - Standard is "Premarket submissions for software contained in medical devices"
  - SILs: "Minor Concern" / "Moderate Concern" / "Major Concern"
- **NASA – US Spacecraft** (NPG 8715.3 Safety Manual)
  - "Catastrophic" / "Critical" / "Moderate" / "Negligible" and probability classes
- **UL – Underwriter Laboratories / Consumer goods**
  - UL 1998 version 2 / May 2000
  - "Critical" / "Non-Critical" / "Supervisory" sections of software
- **FAA – Aircraft safety (Do-178b)**
  - Levels A (catastrophic if faulty) through E (no effect if faulty)
- **US DoD – MIL-STD-882D** (severity below is mapped to 20 risk levels)
  - "Catastrophic" $1M+ / "Critical" $200K+ / "Marginal" $10K+ / "Negligible"
- **Also, NUREGS CRC6463 Software Review Guidelines (Nuclear Power)**
  - Gives design rules & entertaining catalog of common software defects

ENV 50129: Railway Signaling Safety

- **1998 European standards from CENELEC**
  - Covers hardware & overall system; uses SIL
  - Extensive list of techniques:
    - M = Mandatory
    - HR = Highly Recommended
    - R = Recommended
    - NR = Not Recommended
- **Probably more stringent than MISRA**
  - Many years of SW safety
  - Trains can fail safe
  - Trains carry more people

SIL Overview

SAFETY INTEGRITY LEVEL	DEMAND MODE OF OPERATION (probability of failure to perform its design function on demand	CONTINUOUS/HIGH DEMAND MODE OF OPERATION (dangerous failure rate per hour and per element)
4	$< 10^{-7}$	$< 10^{-10}$
3	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-10}$ to $< 0.3\ 10^{-8}$
2	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 0.3\ 10^{-8}$ to $< 10^{-7}$
1	$\geq 10^{-5}$ to $< 10^{-4}$	$\geq 10^{-7}$ to $< 0.3\ 10^{-5}$

[ENV50129]

Example techniques

Technique/Measures	SIL 1	SIL 2	SIL 3	SIL 4
1. Preliminary Hazard Analysis	HR	HR	HR	HR
2. Fault Tree Analysis	R	R	HR	HR
3. FMECA	R	R	HR	HR
4. HAZOP	R	R	HR	HR

[ENV50129]

ENV 50128: Railway Signaling Software Safety

- **2001 European standards from CENELEC**
  - Designed for use with ENV50129
- **List of 69 software safety techniques**
  - Both high level areas & specific techniques
  - Higher SILs require some techniques
  - Some techniques are NR (e.g., artificial intelligence for safety)

TECHNIQUE/MEASURE		Ref	SWS IL0	SWS IL1	SWS IL2	SWS IL3	SWS IL4
1.	Formal Methods including for example CCS, CSP, HOL, LOTOS, OBJ, Temporal Logic, VDM, Z and B	B.30	-	R	R	HR	HR
2.	Semi-Formal Methods	D.7	R	HR	HR	HR	HR
3.	Structured. Methodology including for example JSD, MASCOT, SADT, SDL, SSADM and Yourdon.	B.60	R	HR	HR	HR	HR
4.	Modular Approach	D.9	HR	M	M	M	M
5.	Design and Coding Standards	D.1	HR	HR	HR	M	M
6.	Analysable Programs	B.2	HR	HR	HR	HR	HR
7.	Strongly Typed Programming Language	B.57	R	HR	HR	HR	HR
8.	Structured Programming	B.61	R	HR	HR	HR	HR
9.	Programming Language	D.4	R	HR	HR	HR	HR
10.	Language Subset	B.38	-	-	-	HR	HR
11.	Validated Translator	B.7	R	HR	HR	HR	HR
12.	Translator Proven in Use	B.65	HR	HR	HR	HR	HR

[ENV50128]

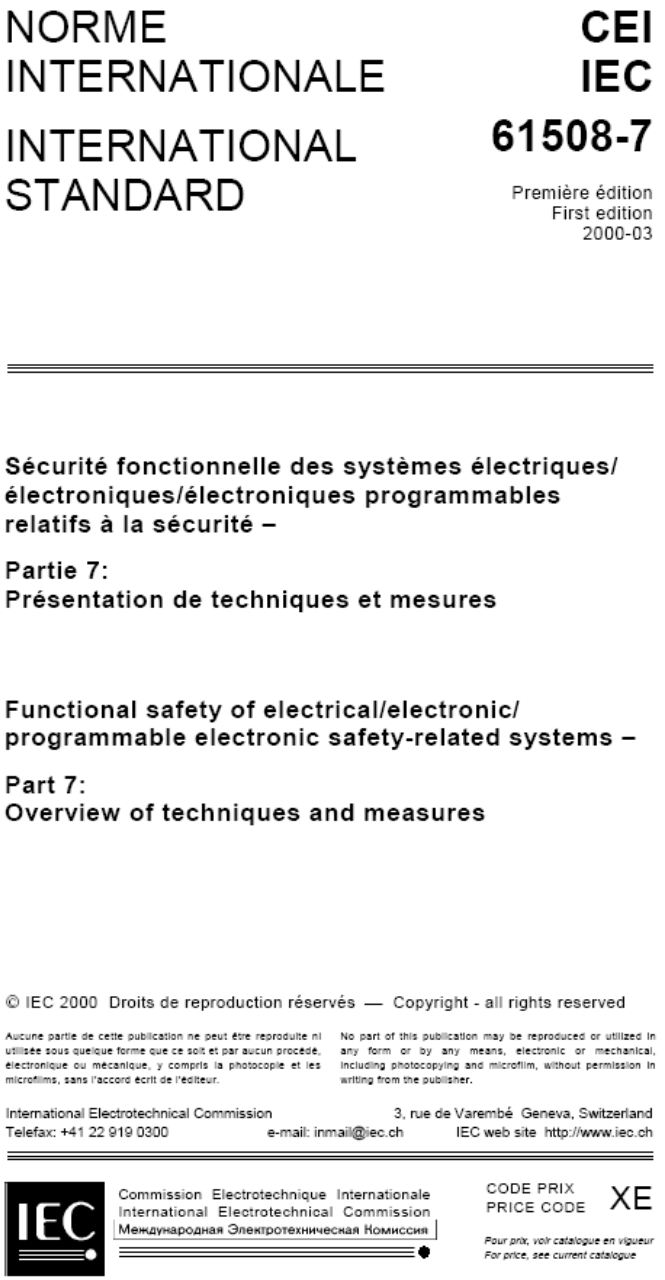
Table A.19 – Static Analysis (D.8)
Referenced by clauses 11 and 14

TECHNIQUE/MEASURE		Ref	SWS IL0	SWS IL1	SWS IL2	SWS IL3	SWS IL4
1.	Boundary Value Analysis	B.4	-	R	R	HR	HR
2.	Checklists	B.8	-	R	R	R	R
3.	Control Flow Analysis	B.9	-	HR	HR	HR	HR
4.	Data Flow Analysis	B.11	-	HR	HR	HR	HR
5.	Error Guessing	B.21	-	R	R	R	R
6.	Fagan Inspections	B.24	-	R	R	HR	HR

[ENV50128]

IEC 61508

- **IEC 61508**
  - The new main standard for software safety
  - Strategy is to tailor for different domains (e.g., chemical process)
  - Comprehensive standard – would be no surprise if MISRA shifts to this for next generation (but, that is speculation on my part)
  - Includes SILs & table of recommended techniques
  - E/E/PES = electrical/electronic/programmable electronic safety-related systems
- **Also, a few new ideas that are useful**
  - E.g., concept of a “proof test”



NORME INTERNATIONALE
INTERNATIONAL STANDARD

CEI IEC 61508-7

Première édition
First edition
2000-03

Sécurité fonctionnelle des systèmes électriques/ électroniques/électroniques programmables relatifs à la sécurité –

Partie 7:
Présentation de techniques et mesures

Functional safety of electrical/electronic/ programmable electronic safety-related systems –

Part 7:
Overview of techniques and measures

International Electrotechnical Commission 3, rue de Varembé Geneva, Switzerland
Telefax: +41 22 919 0300 e-mail: inmail@iec.ch IEC web site http://www.iec.ch

IEC Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

CODE PRIX
PRICE CODE XE

Pour prix, voir catalogue en vigueur
For price, see current catalogue

Parts Of IEC 61508

1. **General Requirements**
2. **Requirements for electrical/electronic/programmable electronic safety-related systems ("E/E/PES")**
   - General hardware/system guidelines
3. **Software requirements**
   - Software-specific techniques
4. **Definitions and abbreviations**
5. **Examples of methods for determination of SILs**
6. **Guidelines on the application of parts 2 & 3**
7. **Overview of techniques and measures**
   - Brief summary description of all techniques mentioned

IEC 61508 Safety Integrity Levels (SILs)

SIL	Continuous/High Demand (Dangerous failures/hr)	Low Demand (Probability of failure to perform on demand)
4	10^{-9} to 10^{-8} /hr	10^{-5} to 10^{-4}
3	10^{-8} to 10^{-7} /hr	10^{-4} to 10^{-3}
2	10^{-7} to 10^{-6} /hr	10^{-3} to 10^{-2}
1	10^{-6} to 10^{-5} /hr	10^{-2} to 10^{-1}

Source: [Redmill98]

- **Continuous / high demand**
  - Continuous operation or very frequent discrete time operation
  - "Critical" boundary is SIL 3 at 10^{-7}/hr
- **Low demand – occasionally activated**
  - For example, emergency backup system
  - Expressed as probability it will fail if it is used one time (e.g., emergency brake)
  - "Critical" boundary is SIL 3 at 10^{-3} – one failure per 1000 activations
  - Presumption is that it is difficult to get into a situation where this is necessary

IEC 61508 Techniques

- **Similar to ENV 50128; with more detail**
  - M = Mandatory
  - HR = Highly Recommended
  - R = Recommended
  - NR = Not Recommended

Table A.1 – Software safety requirements specification (see 7.2)

	Technique/Measure*	Ref.	SIL1	SIL2	SIL3	SIL4
1	Computer-aided specification tools	B.2.4	R	R	HR	HR
2a	Semi-formal methods	Table B.7	R	R	HR	HR
2b	Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR

[IEC 61508-3]

	Technique/Measure*	Ref	SIL1	SIL2	SIL3	SIL4
1	Fault detection and diagnosis	C.3.1	---	R	HR	HR
2	Error detecting and correcting codes	C.3.2	R	R	R	HR
3a	Failure assertion programming	C.3.3	R	R	R	HR
3b	Safety bag techniques	C.3.4	---	R	R	R
3c	Diverse programming	C.3.5	R	R	R	HR
3d	Recovery block	C.3.6	R	R	R	R
3e	Backward recovery	C.3.7	R	R	R	R
3f	Forward recovery	C.3.8	R	R	R	R
3g	Re-try fault recovery mechanisms	C.3.9	R	R	R	HR
3h	Memorising executed cases	C.3.10	---	R	R	HR
4	Graceful degradation	C.3.11	R	R	HR	HR
5	Artificial intelligence - fault correction	C.3.12	---	NR	NR	NR
6	Dynamic reconfiguration	C.3.13	---	NR	NR	NR
7a	Structured methods including for example, JSD, MASCOT, SADT and Yourdon.	C.2.1	HR	HR	HR	HR
7b	Semi-formal methods	Table B.7	R	R	HR	HR
7c	Formal methods including for example, CCS, CSP, HOL, LOTOS, OBJ, temporal logic, VDM and Z	C.2.4	---	R	R	HR
8	Computer-aided specification tools	B.2.4	R	R	HR	HR

[IEC 61508-3]

IEC 61508-7 –Techniques & Measures

- **Dictionary of software dependability techniques**
  - Includes “aim”, “description” and references
  - **178 pages**, with 2 or 3 techniques per page
  - Not all are appropriate in every situation
  - BUT, this is probably the most authoritative + extensive such list
- **Nonetheless, it is incomplete**
  - Does not have much on distributed system dependability
  - IEC 61508 specifically excludes human factors, which is important to vehicles

C.5.15 Fagan inspections

NOTE This technique/measure is referenced in table B.8 of IEC 61508-3.

Aim: To reveal mistakes and faults in all phases of the program development.

Description: A "formal" audit on quality assurance documents aimed at finding mistakes and faults. The inspection procedure consists of five stages: planning, preparation, inspection, rework and follow-up. Each of these stages has its own separate objective. The complete system development (specification, design, coding and testing) must be inspected.

Reference: Design and Code Inspections to Reduce Errors in Program Development. M. E. Fagan, IBM Systems Journal, No. 3, 1976.

ISO 26262 (Automotive Drive-By-Wire)

- **Recommends approaches as in 61508, BUT**
  - Includes human interface
  - Only considers high demand (continuous operation)
  - Claims to be more about normal operation than specific safety functions
    - (but in practice the distinction is blurred even for 61508)

- **ASIL (Automotive SIL)**
  - ISO 26262 part 3 describes ASILs
  - ASIL A (lowest) … ASIL D (highest integrity)
  - ASIL "QM" is equivalent of SIL 0 -- not safety related (Quality Management)
  - ASIL concept is: "use these techniques and it will reduce risk to an acceptable level" (techniques chart similar to 61508)

  - Next slide shows how you get an ASIL

ISO 26262 ASIL Determination

- **Severity:**
  - S0 = no injuries
  - S1 = light and moderate injuries
  - S2 = severe and life-threatening injuries (survival probable)
  - S3 = life-threatening injuries (survival uncertain) and fatalities
- **Probability of exposure:**
  - E0 = Incredible
  - E1 = Very low probability
  - E2 = Low probability (e.g., 1%)
  - E3 = Medium probability
  - E4 = High probability
- **Controllability**
  - C0 = Controllable in general
  - C1 = Simply controllable
  - C2 = Normally controllable (e.g., can control 90% of time)
  - C3 = Difficult to control or uncontrollable
- **Notice that only one bin is ASIL D (highest level)**

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Redundancy and SIL Requirements

- **SIL 1 & SIL 2 are semi-critical – probably nobody dies in mishap**
  - Let's say target is 10^{-5}/hr
  - Single board computer failure rate is ballpark 10^{-5} to 10^{-6}/hr
  - So that means SIL 1 and SIL 2 can be achieved with a single computer

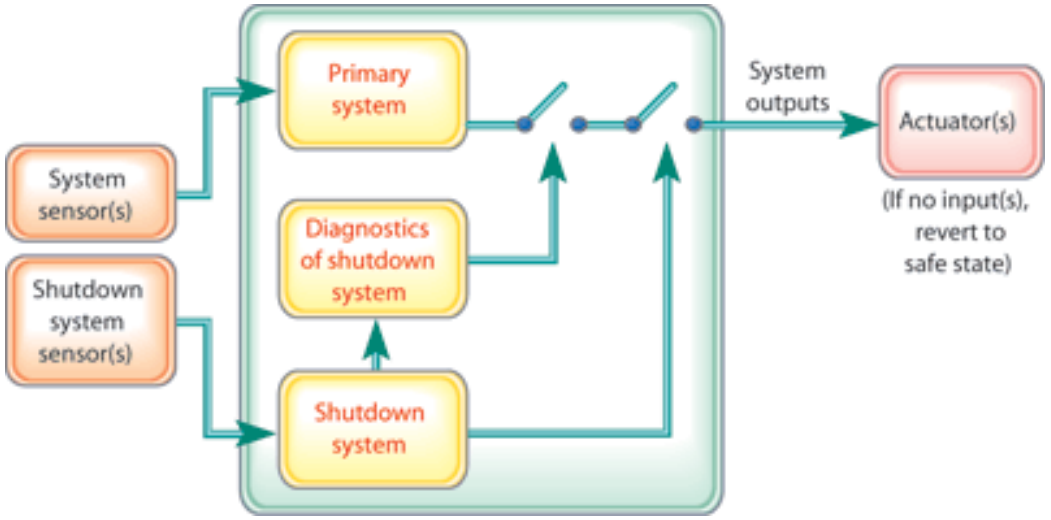
- **SIL 3 & SIL 4 are life-critical – likely that someone dies in mishap**
  - Let's say target is 10^{-7}/hr or lower failure rate
  - This means that you must have *redundancy* to achieve safety
    - In rough terms, two components at 10^{-5}/hr gives you 10^{-10}/hr

  - What failures do you worry about? *Any arbitrary fault*.
    - It's not good enough just to handle faults you can think of
    - Any two functions on a chip can fail together, just because they are on the same chip, even if there is no physical way you can think of for that to happen.
    - As a thought experiment – if you tried your hardest to design an unsafe system on just one chip, would the redundant chip catch it and make it safe? If not, then your design isn't safe.

Example Safe Computing Hardware Patterns

Gated actuation pattern

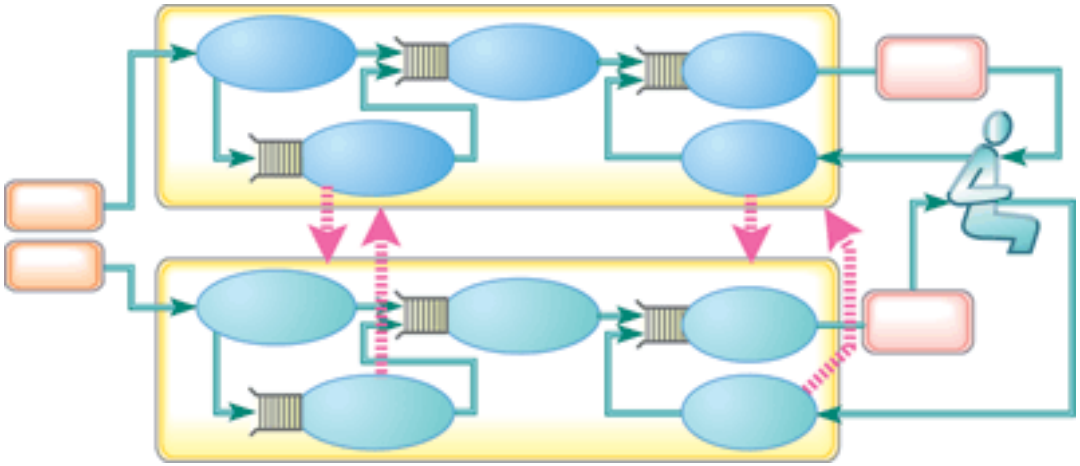
- Variant is a shutdown pattern (resets system if problem)
- Common in automotive systems



[Kalinsky 2005]

Two of Two pattern

- Redundant computation in each CPU shuts down the pair if a mismatch is detected
- Common in Rail systems
- NOT THE SAME as primary/standby, which provides availability but not safe shutdown



Hardware Reliability Matters Too

- **Usually SIL 3 and above require hardware redundancy**
  - Is this single chip CPU good enough for SIL 3? (Possibly not)

MPC564xL Family

Safety with flexibility

Target Applications

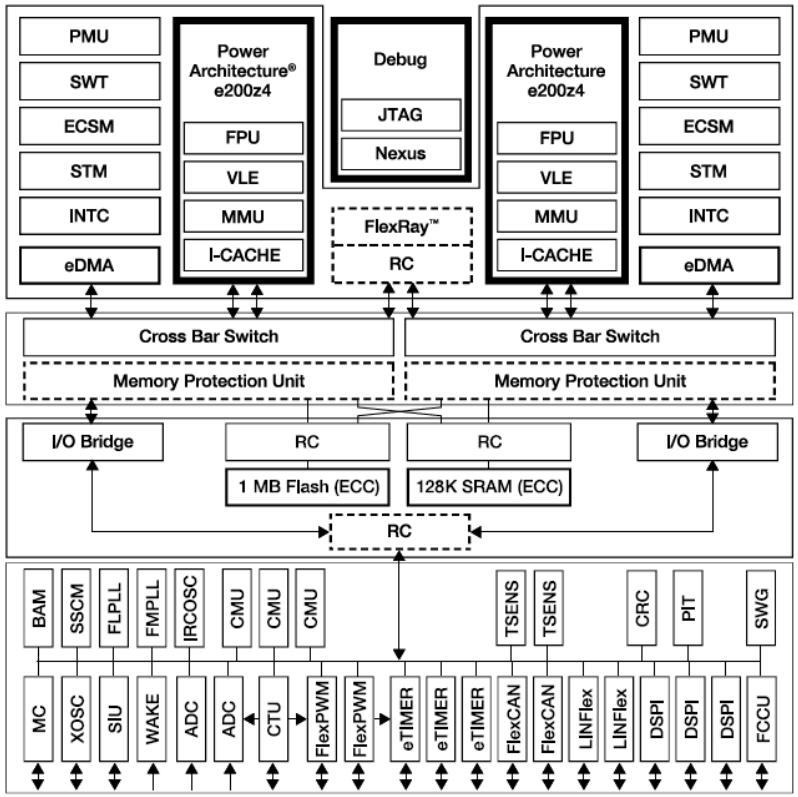
- Electric power steering
- Short- and mid-range adaptive cruise control (up to 100m), RADAR and LIDAR
- Vehicle dynamic and chassis control
- ABS braking systems
- Electrical stability program (ESP)
- Blind spot detection
- Pre-crash detection
- Hybrid electric vehicles

Overview

Designed to specifically address the required IEC61508 (SIL3) and ISO26262 (ASILD) safety standards, the MPC564xL family of microcontrollers takes functional safety, redundancy and cost of ownership to the next level. It reduces design complexity and component count by putting key functional safety features on a single chip with a dual-core, dual-issue architecture.

In lockstep mode it provides an environment for redundant processing and calculations. Common cause failure countermeasures targeting power, clock and error propagation are implemented to detect various failure modes.

MPC564xL Block Diagram



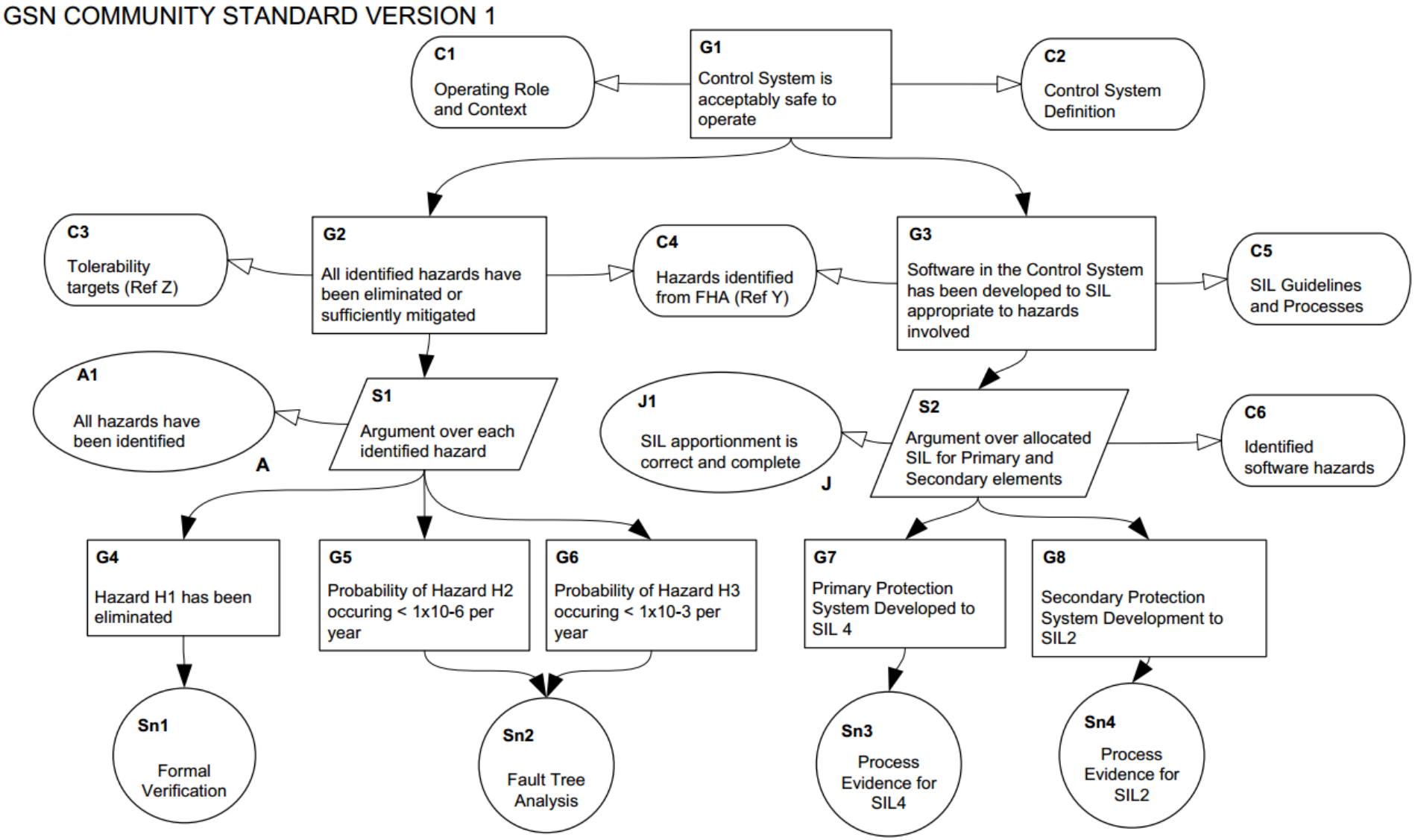
[Freescale09]

freescale semiconductor

Safety Arguments

◆ Goal Structuring Notation (GSN)

http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf



Related Book Chapters

- **Chapter 29 – Watchdog timers**
  - *Should* be in any embedded systems pre-req course
  - But we have found it is often missing, so this chapter catches you up
  - In particular, don't kick the watchdog timer from an ISR
    - Unless the ISR is actually watching the health of every other task in the system!

- **Chapter 30 – System reset**
  - You should always have a way to reset
  - Make sure the system is well behaved during reset
    - What happens if an I/O port hardware defaults to "max speed" upon reset?

Review

MISRA

- Existing standard for automotive industry
- Most prevalent in UK, but spreading to global supplier network
- Probably will be replaced over time by ISO 26262, which is IEC 61508 adapted to automotive industry

IEC 61508

- The new main standard for software safety; background in process industry
  - Strategy is to tailor for different domains (e.g., chemical process)
  - Comprehensive baseline standard (i.e., basis for domain-specific standards)
- Includes SILs & table of recommended techniques

- Does not include human factors as contributor to accidents
  - For automotive, ISO 26262 includes driver as a consideration
- Does not specifically include distributed computing; more a component-level approach