2023

Safety-First

Understanding How To Develop Safety-critical Software

Andreas Weis

C++ now

Safety First!

Understanding How To Develop Safety-Critical Software

Andreas Weis

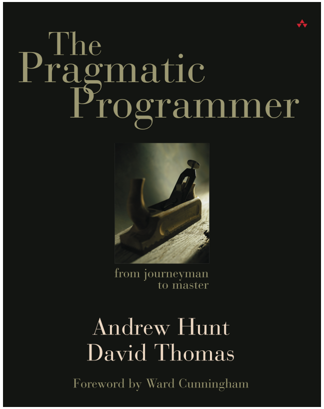
Woven by Toyota

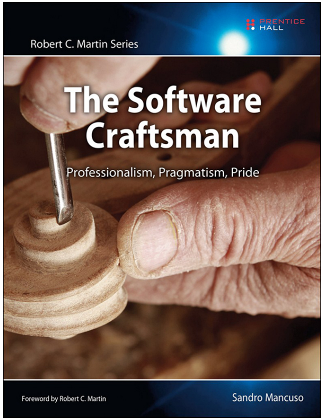
C++Now 2023

About me - Andreas Weis (he/him)

- ComicSansMS
- Co-organizer of the Munich C++ User Group
- Working in automotive on safety-critical components since 2017; member of MISRA C++ WG since 2018 member of ISO SC22 WG21 since 2018;
- Currently working as a Runtime Engineer for Woven by Toyota

Motivation


The Pragmatic Programmer
from journeyman to master
Andrew Hunt
David Thomas
Foreword by Ward Cunningham


PRENTICE HALL
Robert C. Martin Series
The Software Craftsman
Professionalism, Pragmatism, Pride
Foreword by Robert C. Martin
Sandro Mancuso

Fair Warning

This is going to be a very boring talk.

Outline

- What is safety?
- What are the regulations around functional safety
- A tour of an ISO safety standard for software
- Using C++ in a safety-critical environment

What is safety?

Something bad does not happen.[1]

Examples:

- Partial correctness - the program does not produce the wrong answer
- Mutual exclusion - two processes are not in their critical sections at the same time
- Deadlock freedom - The program does not deadlock

[1] Lamport - What Good is Temporal Logic? (1983)

What is safety?

ISO 26262:2018 - Road Vehicles – Functional Safety:

- Harm - Physical injury or damage to the health of persons
- Risk - Probability of the occurrence of harm weighted by the severity
- Unreasonable risk - Unacceptable according to societal moral concepts
- Safety - Absence of unreasonable risk

What is functional safety?

- Electric/electronic systems can fail for various reasons.
  - Systematic faults - Deterministic, need to be prevented by design measures
  - Random hardware failure - Unpredictable failures that follow a probability distribution
- Functional safety wants to reduce the *risk* resulting from such faults
- Note that preventing the fault is just one way to achieve this!

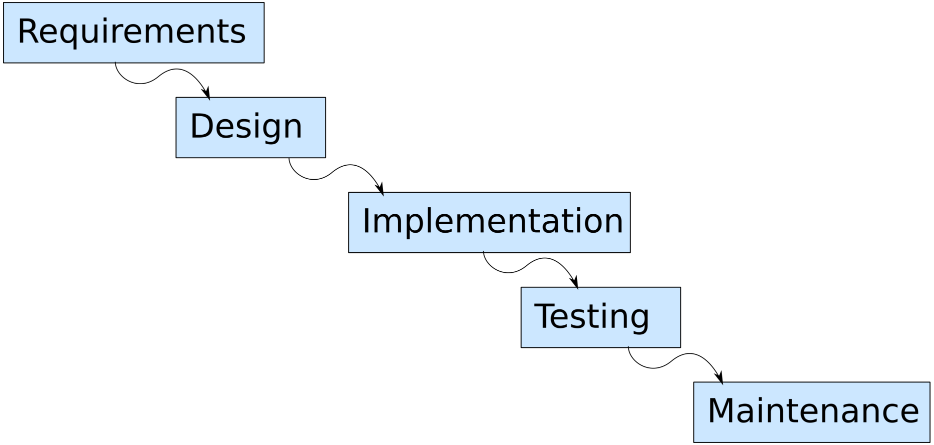
Concerns outside of functional safety

- Quality Management (ISO 9000 family)
- Safety of the intended functionality (ISO 21448:2022)
  - Functional insufficiencies
  - Incorrect handling by the user
  - Functional insufficiencies of AI
- Cybersecurity (ISO/SAE 21434:2021)
  - Attacker exploiting vehicle security vulnerabilities
  - Many security problems are also safety problems
- Specific technologies
  - Eye damage from LIDAR (IEC 60825)
  - AI and Machine Learning (ISO/IEC JTC1/SC42; TR 5469)
  - Cars for drivers with disabilities

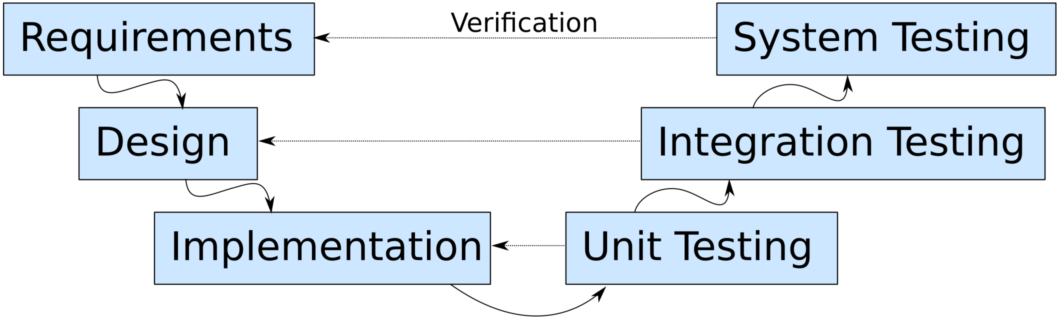
Safety Standards

- IEC 61508 Functional Safety of electrical/eletronic/programmable electronic safety-related systems (1998)
  - ISO 26262 Road Vehicles – Functional safety
  - IEC 62279 Railway
  - IEC 61511 Safety instrumented systems for the process industry sector
  - IEC 61513 Nuclear power plants
  - IEC/EN 62061 Safety of machinery

Waterfall Model



The V-Model of Systems Develoment



Product Life Cycle in ISO 26262

- Based on V-model; V of Vs:
  - Overall system design
  - Hardware development
  - Software development
- Hazard analysis to identify safety goals
- Functional safety concept at system level
- Technical safety concept as input to hardware and software development phases
- Gives exact definitions of outputs and objectives for each phase
- Requires traceability between phases across the entire development

Hazard Analysis and Risk Assessment

- Identify hazardous events on the level of vehicle functions
- Each hazard will have an associated severity and likelihood
- Severity $\times$ Likelihood $=$ Risk
- Risk $\times$ Controllability $=$ ASIL level
- Safety integrity levels: ASIL A, B, C, D; QM
- Outcome: A list of safety goals with ASIL levels

Quantifying Risk - Safety Integrity Levels

- ASIL classification determines the degree of rigor required in addressing the safety goal
- Fulfilling a higher integrity level implies suitability for lower ASIL levels
- ASIL is always associated to a specific safety goal; but the resulting measures may be applicable independently (e.g. when writing a software library)

Functional Safety Concept

Highest level design phase

- Derive from the safety goals a list of functional safety requirements
- Requirements inherit ASIL from the safety goal
- A single requirement may be associated with multiple safety goals

Strategies for Achieving Safety Goals

- Avoid the fault causing a hazard
- Detect the fault and control its impact
- Design the system to be fault-tolerant
  - Timing analysis
  - Redundancy
  - Diversity
- Increase controllability through driver warnings
- Definition of emergency fallback state

Product Development at System Level - Technical Safety Concept

First development phase

- Defines the concrete safety mechanisms that will be responsible for fulfilling the safety requirements
- Technical safety requirements take into account concrete details of the implementation
- Technical safety requirements need to be testable
- Describes functionality of mechanisms including timing and tolerance properties
- Input from the system level into the actual software development process

Software Development Lifecycle

The V inside the V.

- Software safety requirements
- Software architectural design
- Software unit design and implementation
- Unit/Integration/System Testing

Software Safety Requirements

- Derived from the Technical Safety Concept
- Describes how software implements the safety-related functionalities
- Refined Hardware/Software interface
- Optional ASIL decomposition
- Requirements get reviewed extensively
- Basis for software system tests

Software Architectural Design

- Software Design Phase: System must verifiably fulfill the safety requirements
- Software Architecture breaks the system into smaller components
- Describes static and dynamic interaction in a hierarchical structure
- Architects play a vital role in supporting actual verification and implementation efforts

Software Architectural Design

Properties of a good Architecture

- Primary goal is to avoid systematic faults
- Comprehensible, simple, verifiable
- Breakdown into small self-contained pieces
- Clearly defined responsibilities
- Encapsulation of critical data
- Maintainability

Software Architectural Design

Architecture Description

- Write documentation!
- Especially in higher ASIL levels, use semi-formal notation in addition to prose
- Architecture needs to be verifiable
    - Integration tests
    - Simulations on the architecture model

Software Architectural Design

Special concerns of safety software architecture

- Partitioning o software components according to their ASIL level (freedom from interference)
- Analysis of dependent failures
- Analysis to provide evidence for suitability of the design to address safety requirements
- Analysis of resource usage and timing constraints
- Design verification

Safety Element Out Of Context

- Software architecture depends on requirements derived from a specific vehicle function
- Software systems be developed out of context
- All implicit requirements of the software must be documented
- Additional burden on the integrator to ensure that the context in which the system is used is appropriate

Software Unit Design and implementation

- May be auto generated from architecture design model
- Heavily relies on specification form earlier phases
- Main goal is consistency with those specifications
- Simple, readable, comprehensible
- Robust to errors
- Suitable for modification
- Verifiable

Software Unit Design and implementation

Design Principles

- Single-entry , single-exit
- No dynamic objects
- No uninitialized variables
- No global variables
- No variables of same name
- No pointers
- No hidden data flow or control flow
- No recursion

Software Unit Design and implementation

~~[illegible]sign Principles~~

- ~~[illegible]ngle-entry , sin[illegible]~~
- ~~[illegible]ynamic o[illegible]~~
  - ~~[illegible]itia[illegible]iables~~
  - ~~No [illegible]bles~~
  - ~~No v[illegible]f same name~~
  - ~~N[illegible]~~
  - ~~[illegible]dden da[illegible]or control flow~~
- ~~[illegible] recursion~~

Use coding guidelines!

Requirements for programming languages

- Unambiguous and comprehensible definition
  - Formal language spec (also needed for tool qualification)
  - No undefined behavior
  - No footguns
- Support modularity, abstraction and encapsulation
- Support structured programming

Criteria not sufficiently addressed by the language itself shall be covered by coding guidelines.

Coding Guidelines

- Subsetting the language
- Enforcement of low complexity
- Enforcement of strong typing
- Concurrency

MISRA C++

- Discourage use of dangerous language features
- Promote best practices
- Avoid undefined behavior

The guidelines are just one part, the process for enforcing them is just as important.

Following guidelines does not guarantee absence of bugs.

Ensuring Quality - Testing and Verification

- Pair programming and Code reviews
- Static and dynamic code analysis
- (Semi-)Formal verification
- Requirements-based testing
- Fault injection tests
- Resource monitoring

Ensuring Quality - Testing and Verification

Testing methodologies

- Analysis of boundary conditions
- Analysis of equivalence classes
- Analysis of functional dependencies
- Coverage metrics
- Testing on actual target hardware

Testing efforts are continuously monitored and summarized in approved verification reports before production.

Why do we do all this?

Because it works.

Conclusion

- Safety is much more than just the programming language
- Lots of processes and practices
- Most work products from the development are not software
- Software development best practices are the same
- Creation of an environment that ensures those practices are actually followed

Thanks for your attention.

ComicSansMS

https://ghulbus-inc.de/