Zephyr's Roadmap to a Pre-Certified Kernel for Safety-Critical Systems

Dr. Tobias Kästner (inovex)
Embedded World Conference, Nuremberg, March 2026

INNOVATE. INTEGRATE. EXCEED.

embeddedworld
Exhibition&Conference

inovex

inovex

tobias.kaestner@inovex.de
+49 152 3314 8940
@Tobias Kaestner
@tobiaskaestner

Hello,

I am Tobias Kaestner!

Solution Architect Medical IoT at inovex GmbH
#FOSS4MEDICAL

- PhD in Physics (long ago)
- SW/System Architect since 15+ years
  - mainly Medical Devices
- Trainer & Technical Consultant
  - SW-Architecture, Zephyr, Yocto
- In Love w/ Zephyr since 2016
  - realised several prototype projects for life-science R&D
  - currently serving as the project's Safety Architect

inovex

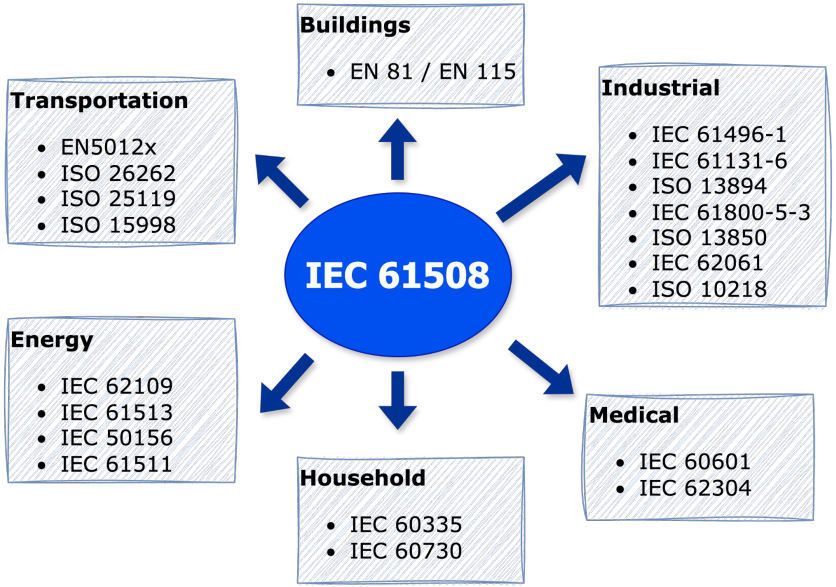
Functional **Safety & Open Source**

Open Source is everywhere ... almost.

- Open Source technologies de-facto choice for non-differentiating software features, driving
  - standardization, collaboration and cost sharing
- Huge number of embedded systems perform safety-critical functions
- Failure of system function can lead to severe injuries or even death of people

inovex

The many standards of Functional Safety.



taken from https://www.tuvsud.com/ko-kr/-/media/global/pdf-files/infographics/tuvsud-functional-safety-in-a-nutshell-infographic.pdf

inovex

Functional Safety & FOSS - The good, the bad & the ugly.

More and more examples where FOSS aims to enter the safety-critical domain

- **Zephyr**
- XEN Hypervisor
- ELISA (Embedded Linux in Safety Applications)
- RTEMS
- Eclipse ThreadX

Adopting FOSS for safety-critical systems faces severe challenges:

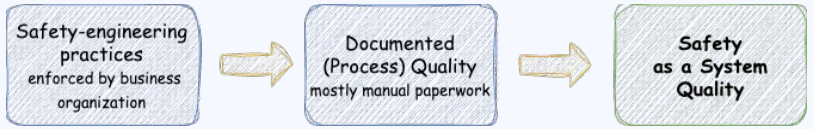
- **Non-free standards restrict participation:** This includes most ISO/IEC/EC standards and MISRA Coding Guidelines.
- **Safety isn't a priority for all FOSS stakeholders,** unlike security.
- **Safety standards focus on the development process,** which must be tailored to business processes.

inovex

The Open Source Dilemma: Contribution-Driven vs Requirements-Driven.

Safety Standards are Process Standards

- Assume enforcement by business owners (liability)
- However, FOSS projects have a governance structure (at best)
  - have control over contribution guidelines to reject unsuitable work
  - but **no way to mandate "required" work to happen**



?

Contribution-Driven (The Power of Open Source)

- Bottom-up, dynamic, fast-paced
- Prioritizes innovation and community participation

Requirements-Driven (The Mandate of Functional Safety)

- Top-down, methodical, favors stability
- Prioritizes rigor, predictability, and auditable proof

inovex

Safety Engineering for Software

inovex

Functional Safety is more than just good code.

Functional Correctness

reliable & dependable,
no defects
all tests green

Active Safety

hazard detection &
protection, safe-states are
known and entered

Reasoning

arguments can be made that
safety was a key design
concern backed by evidence

What happens when things go wrong?

"Absence of Unreasonable Risk** From **Hazards** Caused by **System Malfunctions"

inovex

No Safety without Security.

Security:

Protect machines from (maliciously acting) humans

Safety:

Protect humans from machines going wild

- Most devices are connected by now
- Insecure systems un-safe, too
  - e.g. attackers could nullify safety measures to harm people
- Yet, securing systems may introduce safety risks
  - e.g. FOTA updates to mitigate CVEs

https://www.hipaajournal.com/security-issues-identified-in-75-of-infusion-pumps/
https://www.ahajournals.org/doi/10.1161/CIRCULATIONAHA.118.037331

inovex

Why things break - Cosmic rays and Human Error.

Distinction in standards like **IEC 61508** and **ISO 26262**

Random Hardware Failure

- probabilistic in nature & unpredictable (memory bit-flip from radiation, physical disconnect of a sensor)

Manage by Architectural solutions

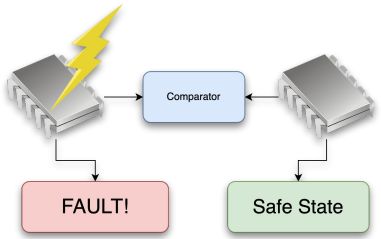
- redundancy, error detection, memory protection, watchdogs

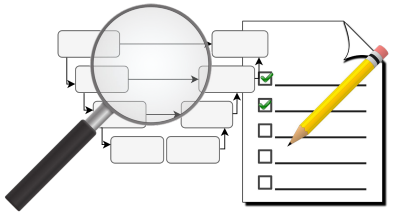
Systematic Failures

- deterministic faults in the design/code (bugs)

Minimize by rigorous, methodical processes

- traceability, coding standards, verification





inovex

Demonstrating **Systematic Capability**

inovex

Zephyr's way towards systematic capability.

IEC 61508-3, Clause 7.4.2.12

“Where a **pre-existing software element** is **reused** to implement all or part of a safety function, the element shall meet both requirements a) and b) below for systematic safety integrity:

a) Meet the requirements of one of the following compliance routes:

- Route 1s: compliant development. Compliance with the requirements of this standard for the avoidance and control of systematic faults in software;
- Route 2s: proven in use. Provide evidence that the element is proven in use. See 7.4.10 of IEC 61508-2;
- **Route 3s**: assessment of non-compliant development. Compliance with 7.4.2.13

inovex

Zephyr's way towards systematic capability.

IEC 61508-3, Clause 7.4.2.13

“To comply with Route 3s a pre-existing software element shall meet all of the following **requirements** a) to i) … “

- Providing a **safety scope definition**
- Creating **requirements** & establishing **traceability** to code & tests
- Creation of **system- & software specification**
- Definition of the **safety claims**
- Using the existing tests, establishing traceability & enhancing coverage
- Creation of the **safety manual**

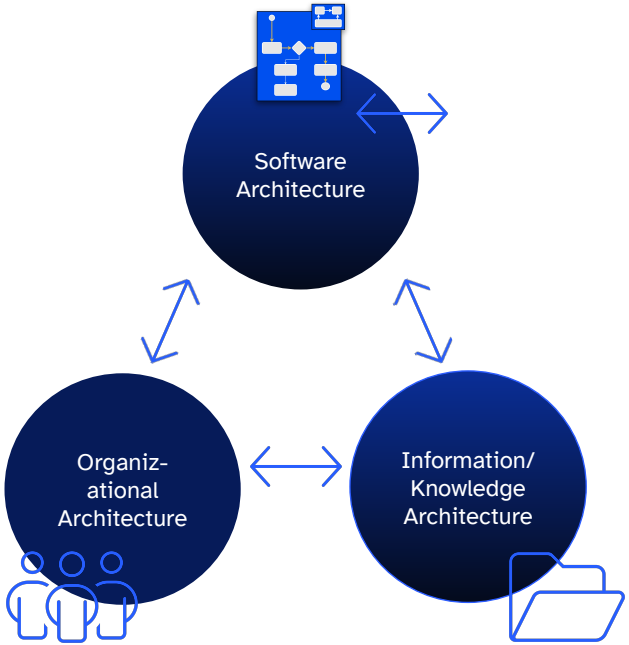
inovex

Functional Safety's profound impact.

Software creation is a **socio-technical** endeavour
Interdependencies between

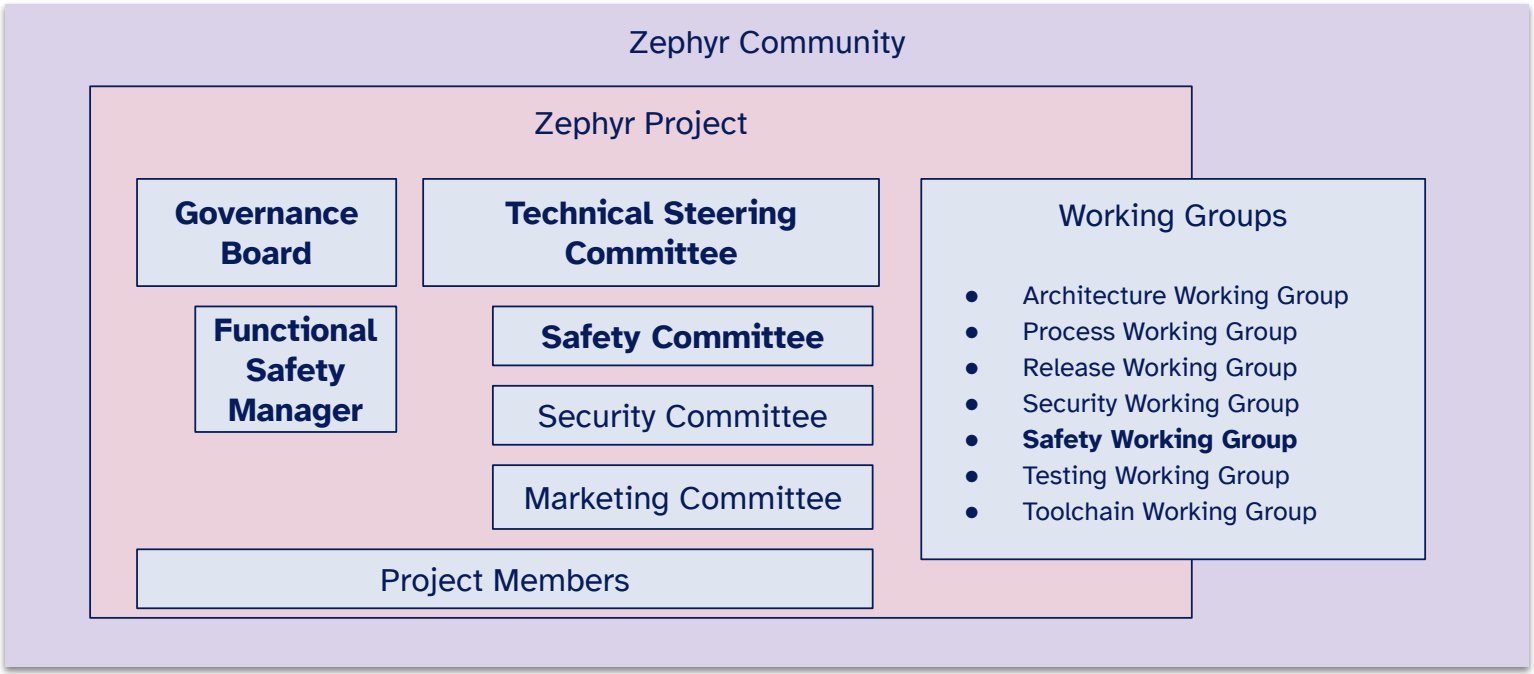
- The organization creating the software
- The way information is recorded & disseminated
- The software's actual architecture & design

To demonstrate **systematic capabilities**
requires adaptation of all three



inovex

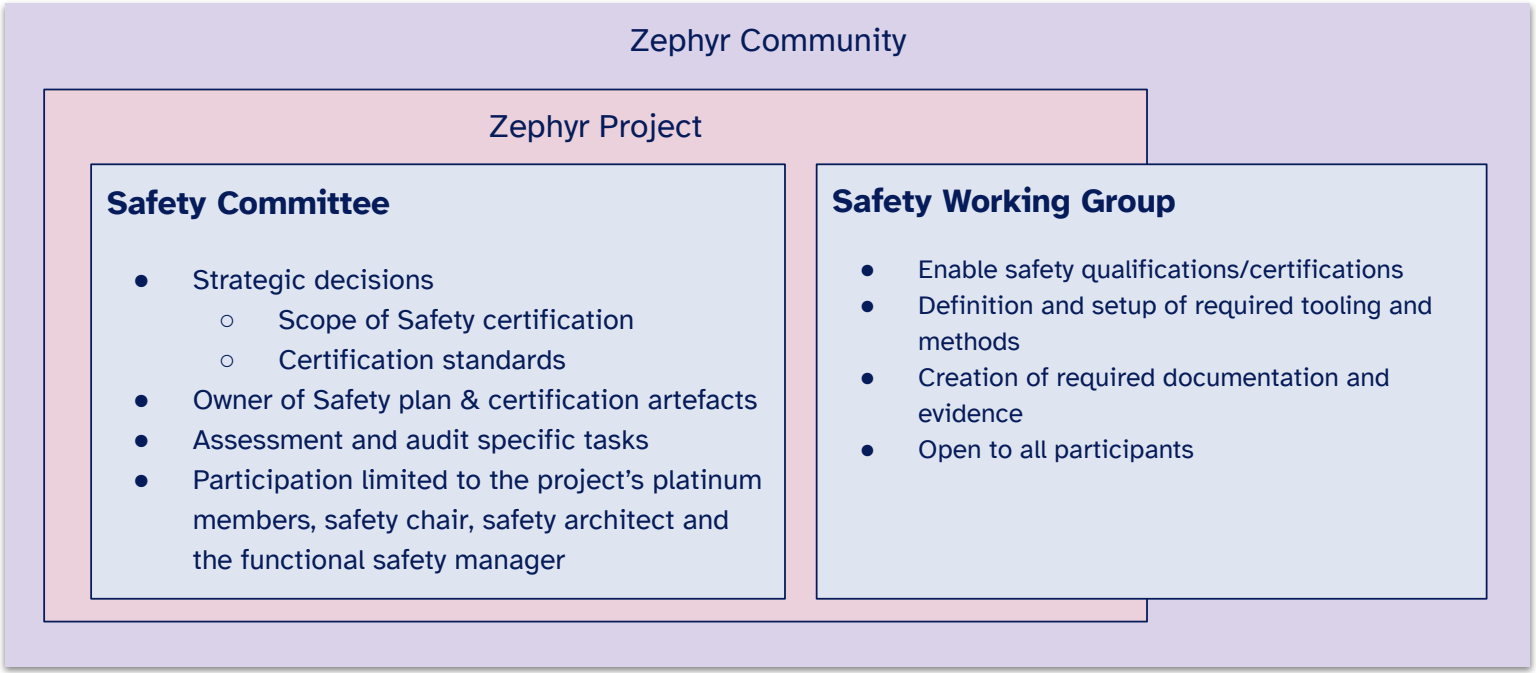
A Safety View on Zephyr's Project Governance.



https://github.com/zephyrproject-rtos/zephyr/wiki/Zephyr-Committee-and-Working-Groups

inovex

A Safety View on Zephyr's Project Governance.



https://github.com/zephyrproject-rtos/zephyr/wiki/Zephyr-Committee-and-Working-Groups

inovex

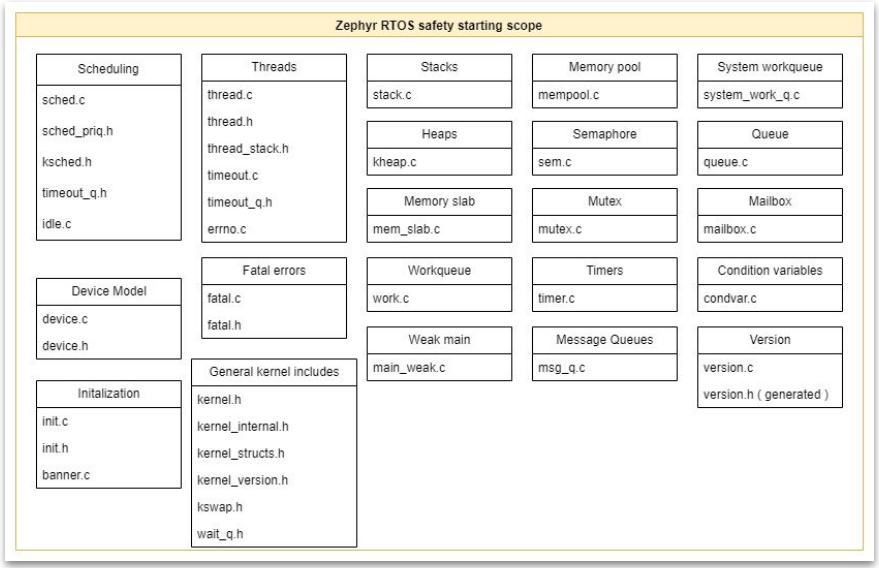
Zephyr's Safety Scope.

Certifying all of Zephyr is infeasible!

Focus for Safety is on

- System Initialization
- Kernel Services & Kernel API
- selected safety-related Subsystems, ie. Task Watchdog

15k LoC out of 2.4M LoC



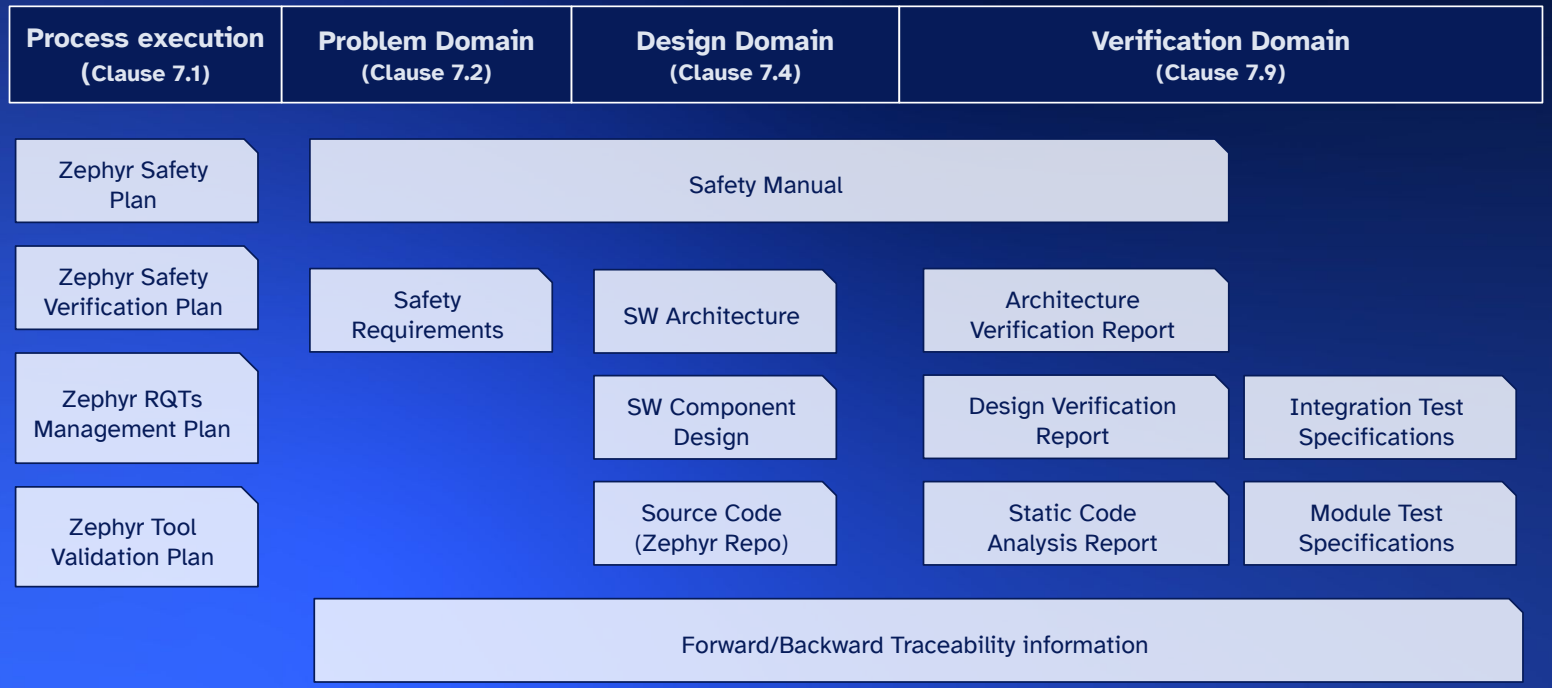
Discussion and current consensus
https://github.com/zephyrproject-rtos/zephyr/issues/57702

inovex

Safety Artifacts & Evidence

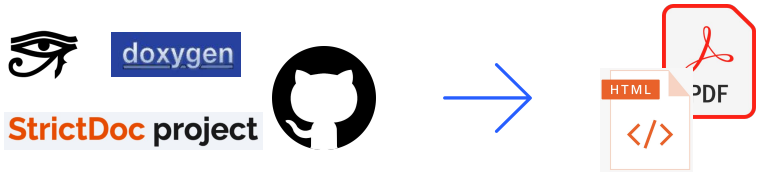
inovex

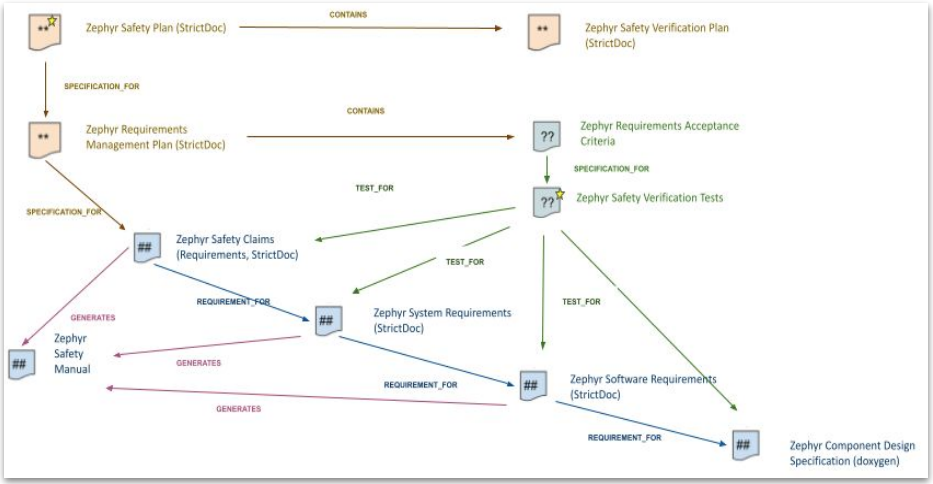
Compiling the evidence for the safety scope.



inovex

Compiling the evidence for the safety scope.





Tools & Methods

- need to be compatible with FOSS practices and free to use
- Developer-friendly tooling
- Docs as Code
  - keep specifications as close to actual code as possible
  - ensure consistency with actual code base
- Version control everything through git
- Automation through integration with Zephyr's Github workflows

inovex

Zephyr as Safety Element out of Context.

Provider

must provide rules how to integrate

Integrator

must adhere to integration rules

inovex

Zephyr's Architecture
for safe and secure systems

inovex

Freedom from interference - A typical example.

Hazards

A low-prio task enters an infinite loop/monopolizes the CPU, preventing high-prio safety-critical task (brake controller) to execute on time

A faulty pointer in a non-critical task writes outside of its own memory area, corrupting the data or stack of a safety-critical task, causing it to behave erratically or crash.

Safety Requirement (no temporal interference)

The SW shall guarantee that a high-priority task can always preempt a lower-priority one to meet its deadline.

Safety Requirement (no spatial interference)

The SW shall enforce strict memory boundaries between software components, ensuring one cannot corrupt another.

pre-emptive, priority-based task scheduler

RTOS responsibilities

HW (MPU) assisted memory partitioning and exception handling

inovex

Freedom from interference - Zephyr's solution.

Safety Requirement (no spatial interference)

The RTOS shall enforce strict memory boundaries between software components, ensuring one cannot corrupt another.

Architectural Solutions:
HW assisted memory partitioning and exception handling,
KConfig for SW Feature management, CMake build system, System Call semantics

Assumptions:
ARCH_HAS_USERSPACE
ARCH_HAS_STACK_PROTECTION

KConfig Feature:
USERSPACE,
HW_STACK_PROTECTION

Build System:
`Linker scripts, Linking stages`
`Features excluded by default`

Safety Application Configuration:
CONFIG_USERSPACE=y
CONFIG_HW_STACK_PROTECTION=y

Tooling:
kconfiglib,
gen_sycalls.py, gen_kobject_list.py
and many more

Source Code:
#ifdef CONFIG_USERSPACE ...

Safety Application Design:
Only enable required features
Tasks as user mode threads,
Safety functions in highest priority threads

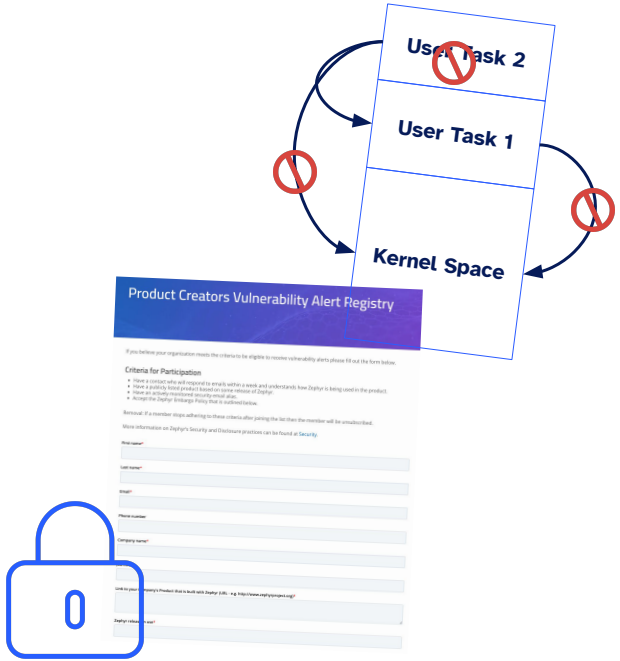
Documentation & Verification required

Obligations by the Safety Manual

inovex

Additional Safety/Security Features available today.

- Error handling and safe-state handlers
- No dynamic memory usage inside kernel
- APIs to assign all Kernel objects memory at link time
- Watchdogs
- Crypto API for secure communication
- TF-M supported for Cortex-M incl. secure boot
- Vulnerability Alert Registry and CVE Numbering Authority



inovex

Current Status & Outlook

inovex

Achieved Milestones.

- **Safety Plans in place**

  Initial set of plans created, reviewed and submitted to TÜV Süd

- **Coding Guidelines**

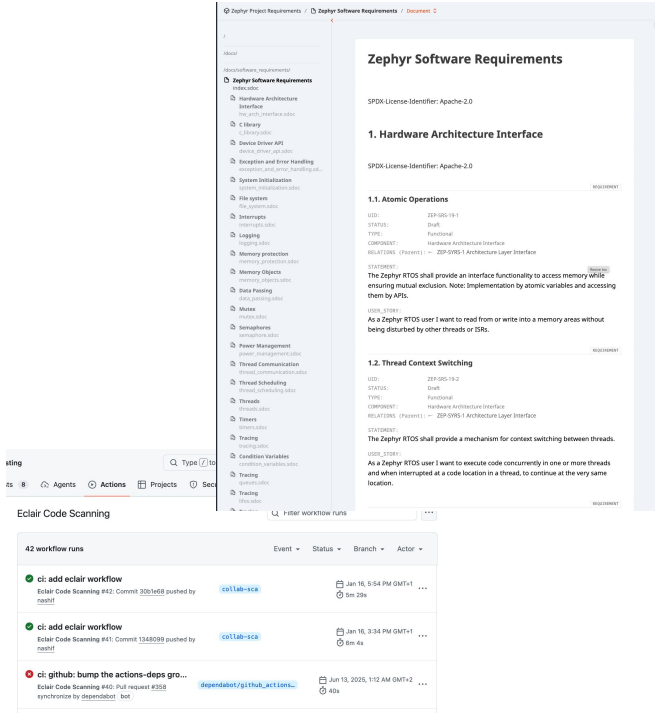
  Defined Zephyr-specific MISRA-C 2012 subset; SCA integrated into Zephyr Build system; automated check integration in progress.

- **Requirements Repository Established**

  https://github.com/zephyrproject-rtos/reqmgmt
  https://zephyrproject-rtos.github.io/reqmgmt/

- **TÜV Concept Approval**

  TÜV Süd confirmed Zephyr's certification concept is feasible.



Key to success: Involving the community

inovex

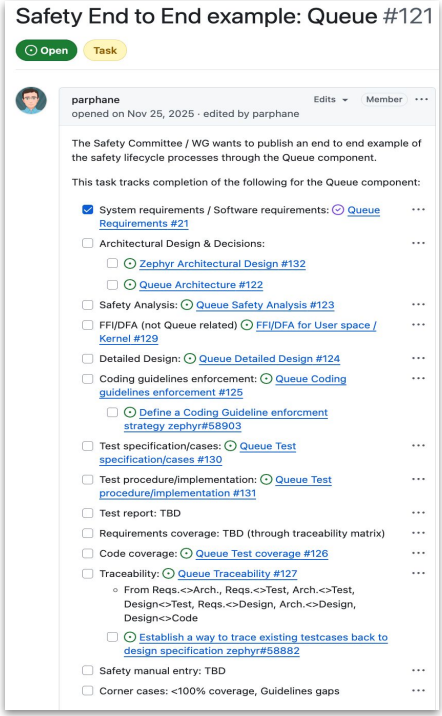
Overall Roadmap and next milestones.

01 Create organizational structures
- Safety Committee, Safety-WG, Budget
- Safety Chair, Safety Architect...

02 Create End to End working model
Define the Safety Scope
- Methods, Content, Tooling
- Kernel API, Arch independent code

03

04 Compile the evidence
- It's done when it's done



https://github.com/zephyrproject-rtos/regmgmt/issues/121

inovex

But Your Certificate is Already Late - Towards Continuous Certifiability.

Zephyr is ever evolving: ~6,000 commits/release (40k commits from 2.7 to 3.7)

- **So is the Safety Scope:** avg 60 commits/release (500 commits from 2.7 to 3.7)
- **Security mandates timely updates, too - And no safety without security**

What happens for Safety must (eventually) happen on main

inovex

But Your Certificate is Already Late - Towards Continuous Certifiability.

Ideal situation (from a Safety & Security perspective):

- Safety documentation remains in sync with **main** (for Safety scope)
- Security updates provably not interfering with Safety claims
- Re-Certification of major (LTS) releases become no-brainers

Safety Working model must accommodate for this

- need to find right balance between innovation and stability for Safety scope
  - additional compliance checks
  - additional manual reviews for PRs affecting Safety scope
- Automate “everything”: Static Code analysis, Traceability Checks, Document generation

No prior experiences - Zephyr is at the frontier of known procedures

Want to make it happen faster? Join the project!

Initial target is **IEC 61508 SIL 3 / SC 3** (with Option for 26262 ASIL D)

- **Zephyr** to be treated as **Safety Element out of Context** (SEooC)
- Limited scope of kernel functions and interfaces similar to other RTOSs

Safety Working Group

subscribe	**Mailing List**	https://lists.zephyrproject.org/g/safety-wg
chat	**Discord**	https://discord.gg/mgZkSmq2
meet	**Video Call every 2nd Tuesday 4pm CET**	

Hall 4
Booth 170

Thank you!

Tobias Kaestner

Solution Architect Medical IoT at inovex GmbH

+49 152 3314 8940

tobias.kaestner@inovex.de

inovex