Zephyr™

Developing Open-Source Software RTOS with Functional Safety in Mind ?

Anas Nashif, Intel Open-Source Technology Center

Disclaimer:

I am not a safety expert.

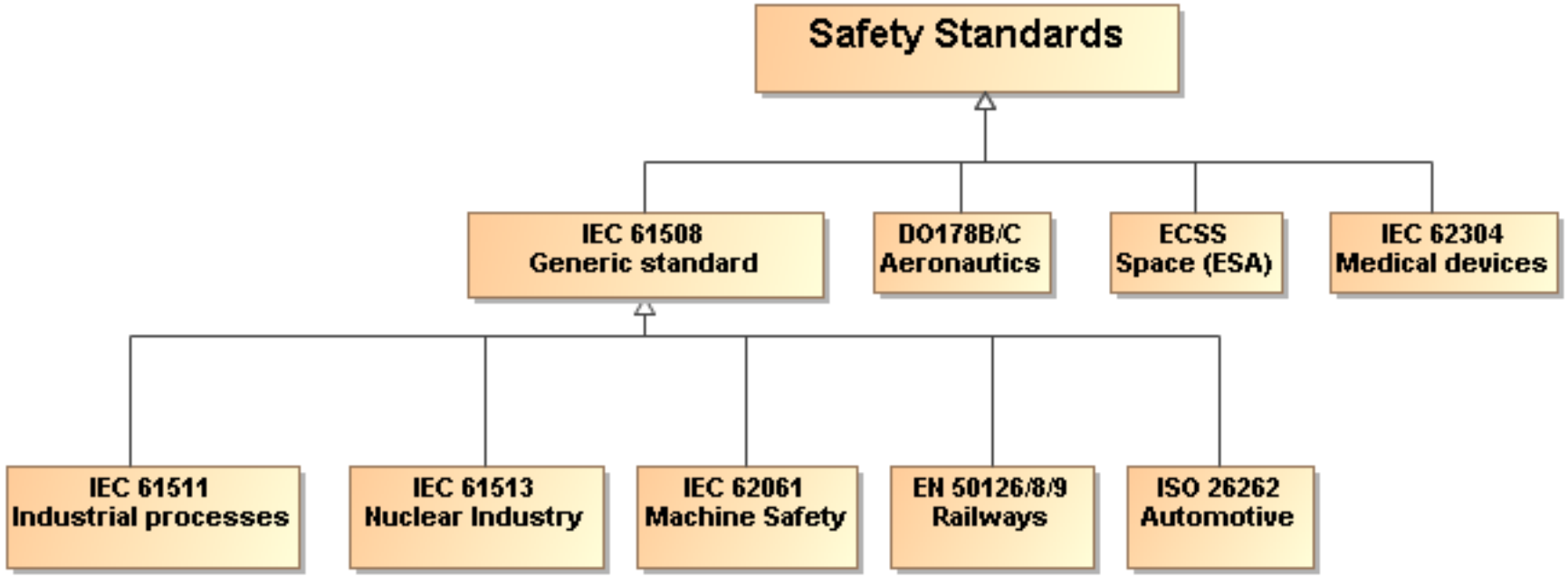
Credit: https://unsplash.com/@rawpixel


Browse
Now Playing
i9:zephyr(master): cd
i9:~: git clone git@github.com:zephyrproject-rtos/zephyr.git
Cloning into 'zephyr'...
remote: Enumerating objects: 10, done.
remote: Counting objects: 100% (10/10), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 260365 (delta 5), reused 4 (delta 4), pack-reused 260355
Receiving objects: 100% (260365/260365), 148.41 MiB | 3.87 MiB/s, done.
Resolving deltas: 100% (189420/189420), done.
Checking out files: 100% (11912/11912), done.
i9:~:

Can open-source software be used for Functional Safety?

- Short answer is yes, using open-source software is very common, in all areas
- But…
  - Open-source software usually require major transformation before it can be used
  - Mostly such transformation happens behind closed doors (if license allows that)
  - Complete disconnect between original source and “certified” code
  - Transformation of open-source code to be functionally safe is “expensive”
  - Following standards very early in a project life-cycle is key
  - There are many standards…

Safety Standards



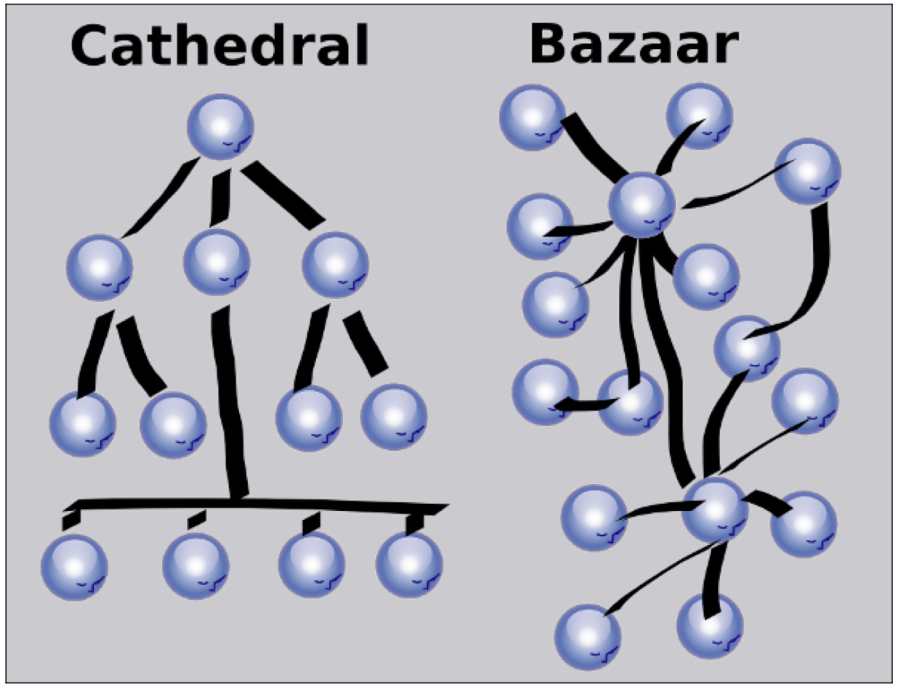
"The nice thing about standards is that there are so many of them to choose from." [Tanenbaum]

Is this possible? Example: RTOS

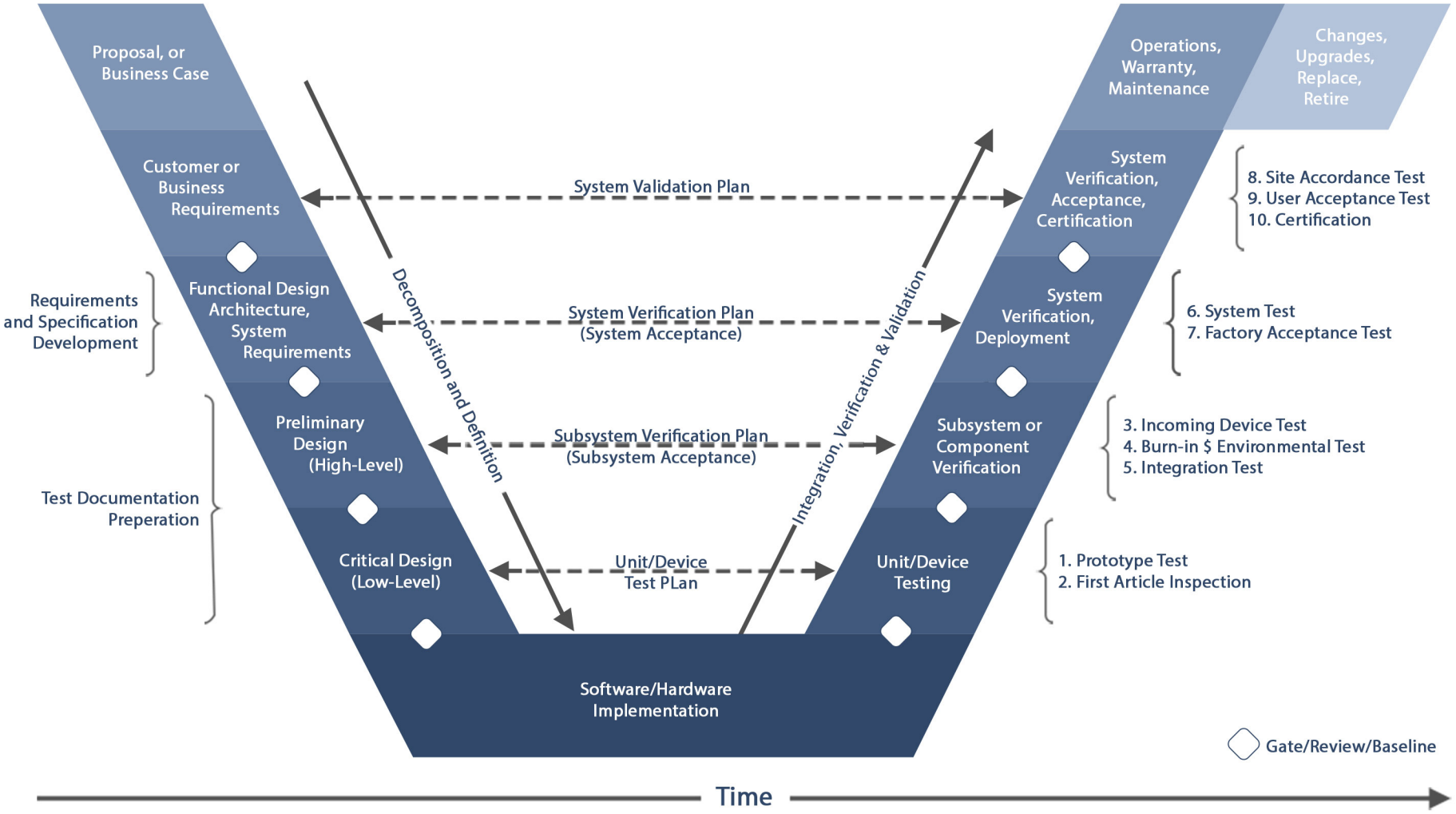
- Open source implementation
- Small trusted code base (in terms of LoC)
- Safety oriented architecture
- Built in security model
- POSIX compliant C library
- Supports deterministic thread scheduling
- Supports multi-core thread scheduling
- Proof that ISO compliant development was done
- Accountability for the implementation
- Industry Adoption
- Certification friendly interfaces

Cathedral and the Bazaar

- Open-source Software is not a problem in itself
- It is difficult to map a stereotypical open-source development to the V-model
  - Specification of features
  - Comprehensive documentation
  - Traceability from requirements to source code
  - Number of committers and information known about them
  - Certification authority not familiar with open-source development

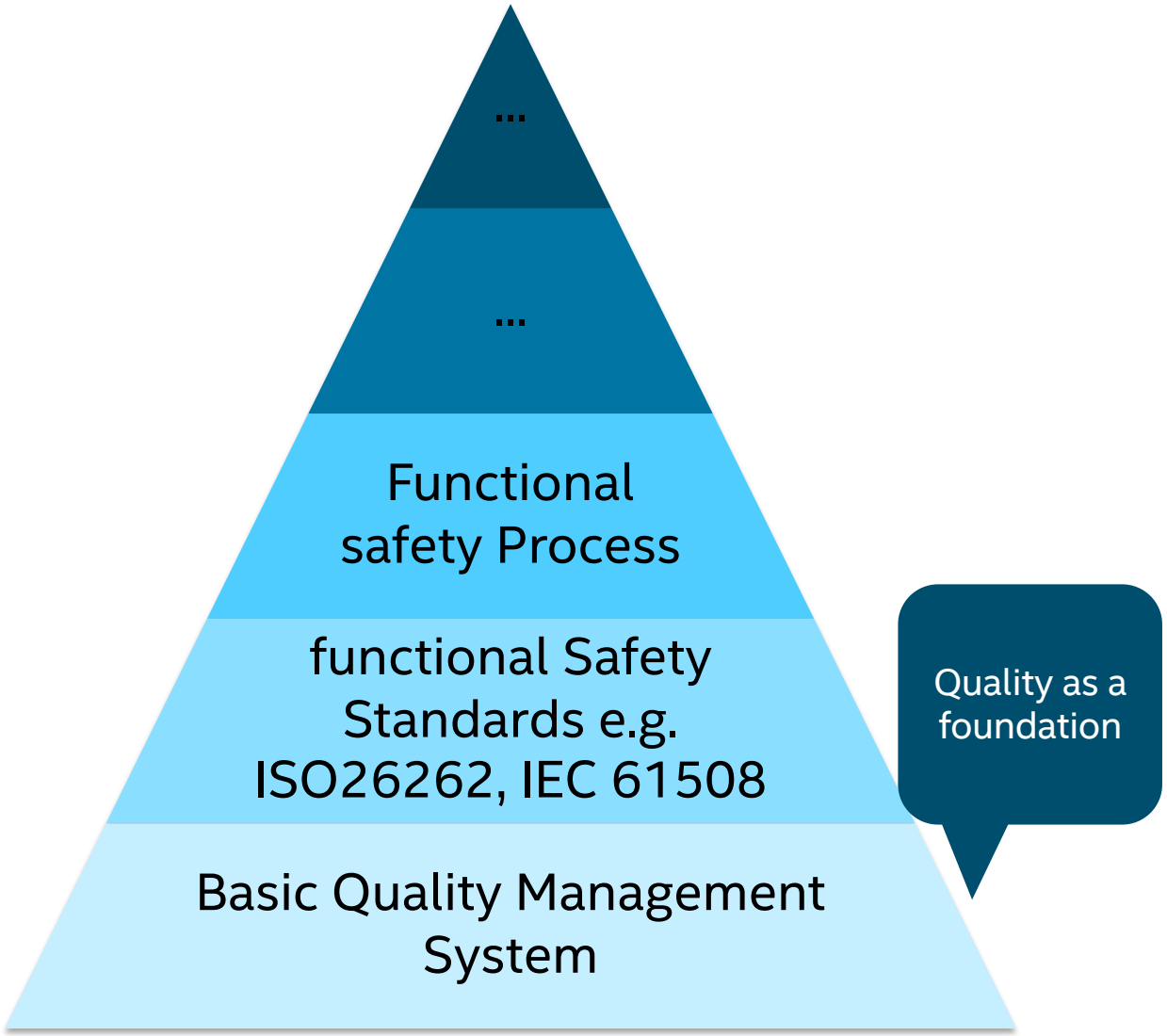


V-Model: Software Development



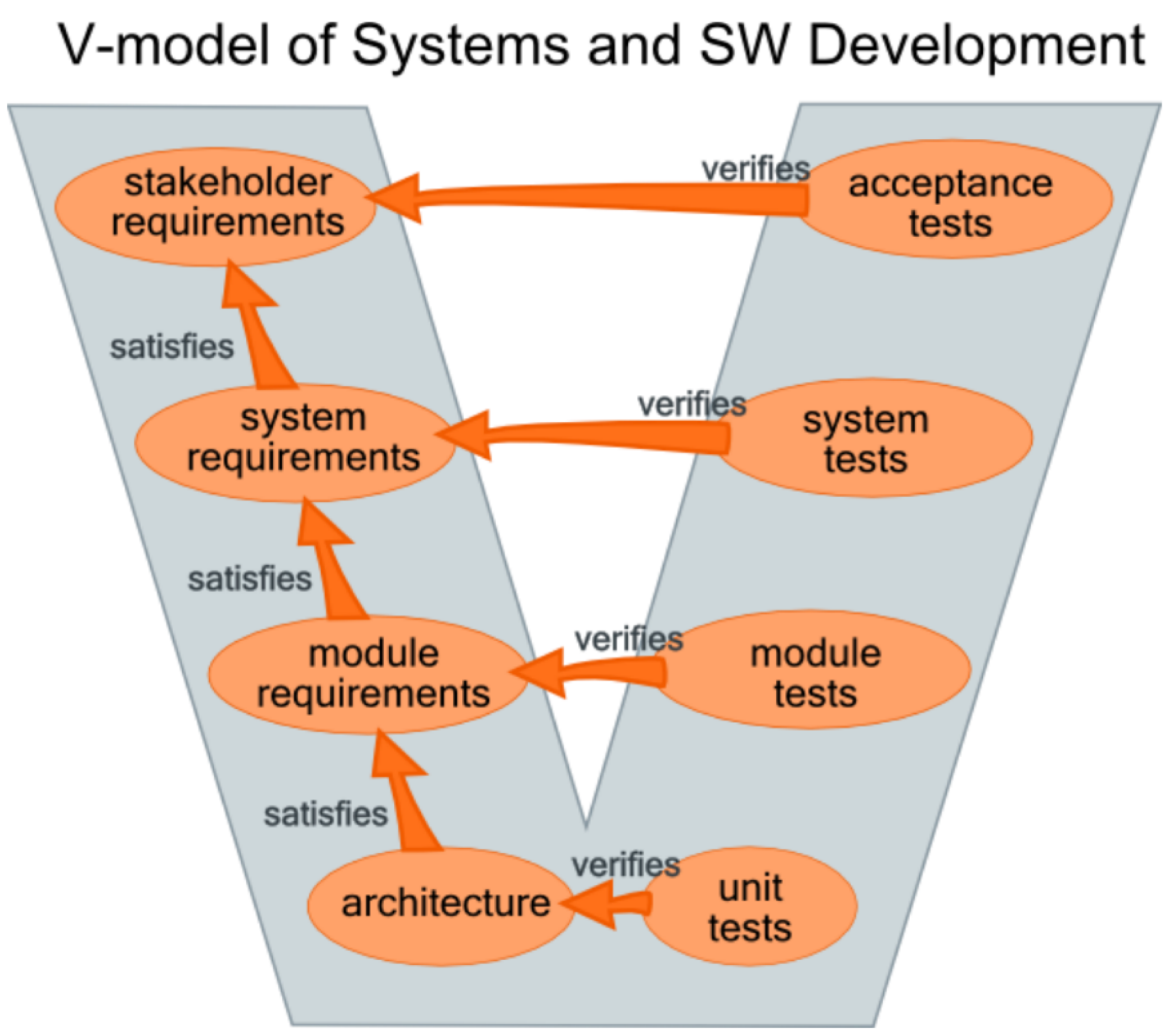
Quality Matters

- Quality is a mandatory expectation for software across the industry.
- Software Quality is not an additional requirement caused by functional safety standards.
- Functional safety considers Quality as an existing pre-condition.
- Quality Managed (QM) status should be the aspiration of any open-source project, regardless of FuSA goals

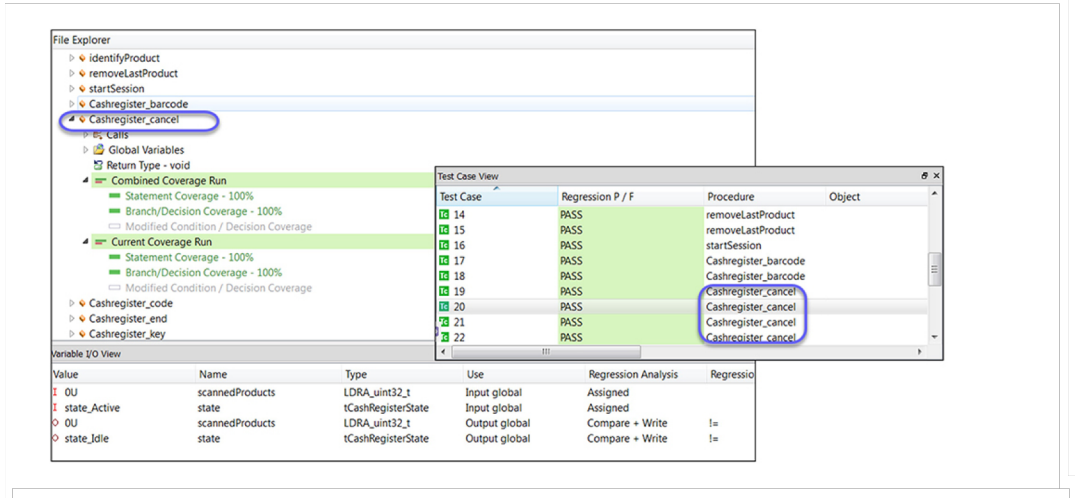


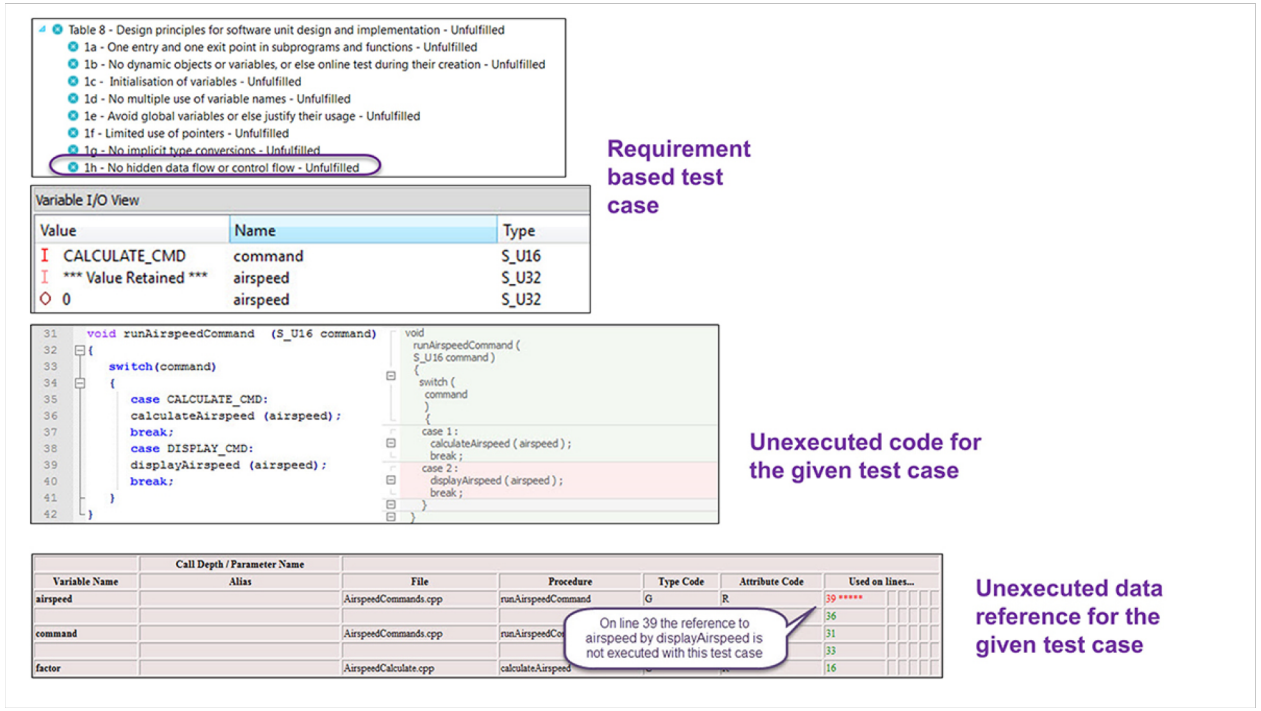
Requirement Traceability

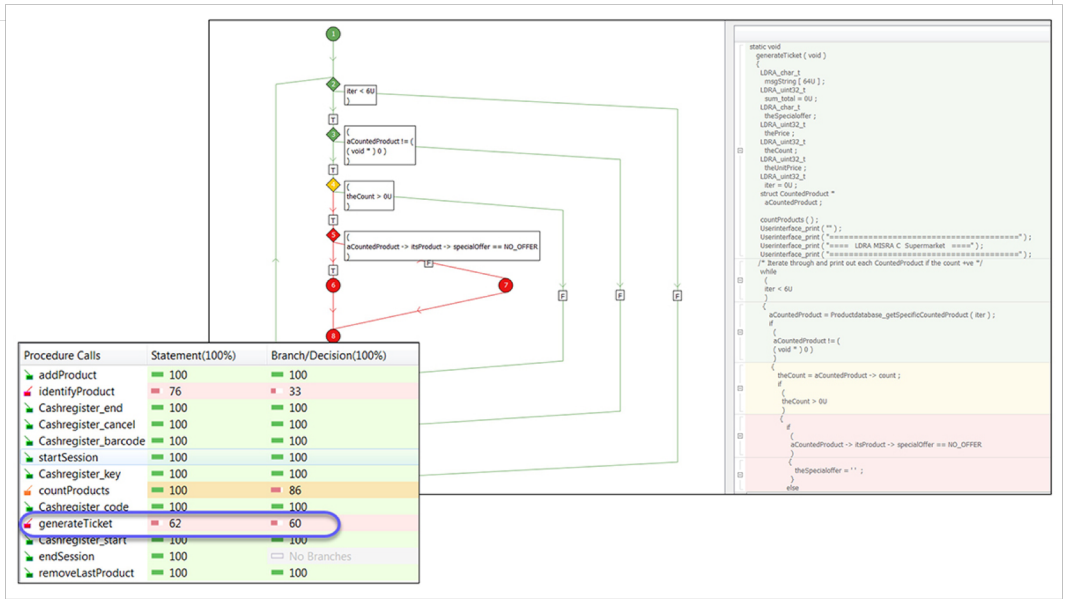
- Reference links between requirements
- Verification links from related tests
- Satisfaction links from decomposec requirements
- Implementation links from user stories

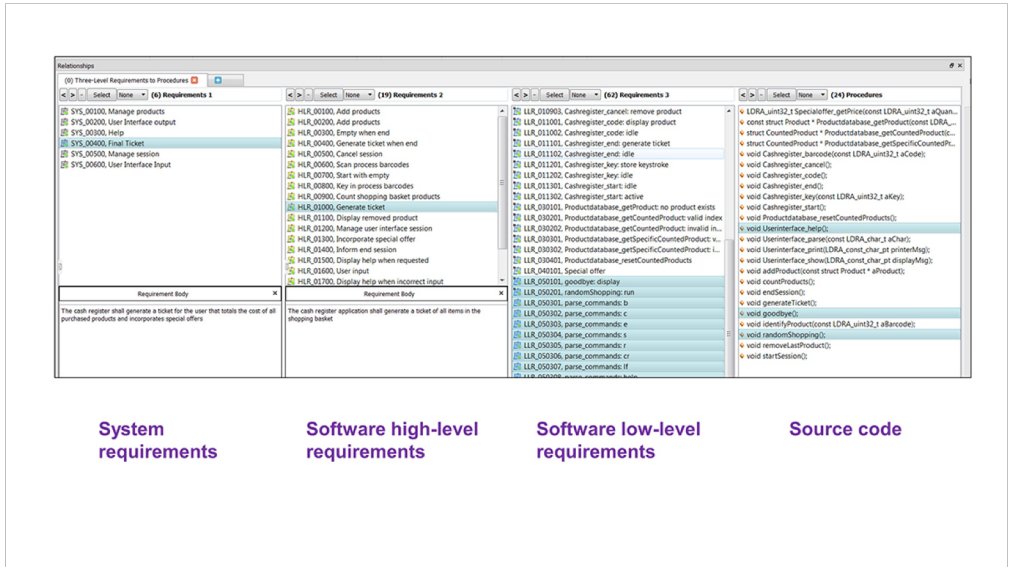


Traceability Tools









MISRA-C as a Guideline

- It is a software development standard that aims to facilitate programming safety-critical software in embedded systems
  - Focus in safety, security, portability and reliability.
- Latest version is MISRA-C:2012
  - Launched in 2013 and it's based on ISO/IEC 9899:1999.
- Contains 167 guidelines in the standard plus 14 new guidelines in Amendment 1
- Every MISRA C guideline is classified as either being a "rule" or a "directive".
  - A directive is a guideline that is not possible to provide the full description necessary to perform a check for compliance.
  - A rule is a guideline for which a complete description of the requirement has been provided, it is possible to check compliance without needing any other information.
- A guideline can be "mandatory", "required" or "advisory"
  - **Mandatory** - All code shall comply with every mandatory guideline. Deviation is not permitted.
  - **Required** - All code shall comply with every required guideline. Deviation is allowed.
  - **Advisory** - It is a recommendation. Formal deviation is not necessary.

MISRA-C and Opensource Challenges

- Some Rules are very controversial, how to deal with those?
- Decide which guidelines you want to deviate
- Incorporate it to contribution guidelines
  - MISRA-C is proprietary, how to make it available for everybody
- Find the right “opensource” tools and integrate with CI
  - Most tools are commercial, not easy to integrate on Github with PRs
- Collaboration from other developers
  - Either, reviewing and fixing
- Apply it to the full scope of a project.

Example: MISRA-C Rule 15.5

Rule 15.5 - A function should have a single point of exit at the end

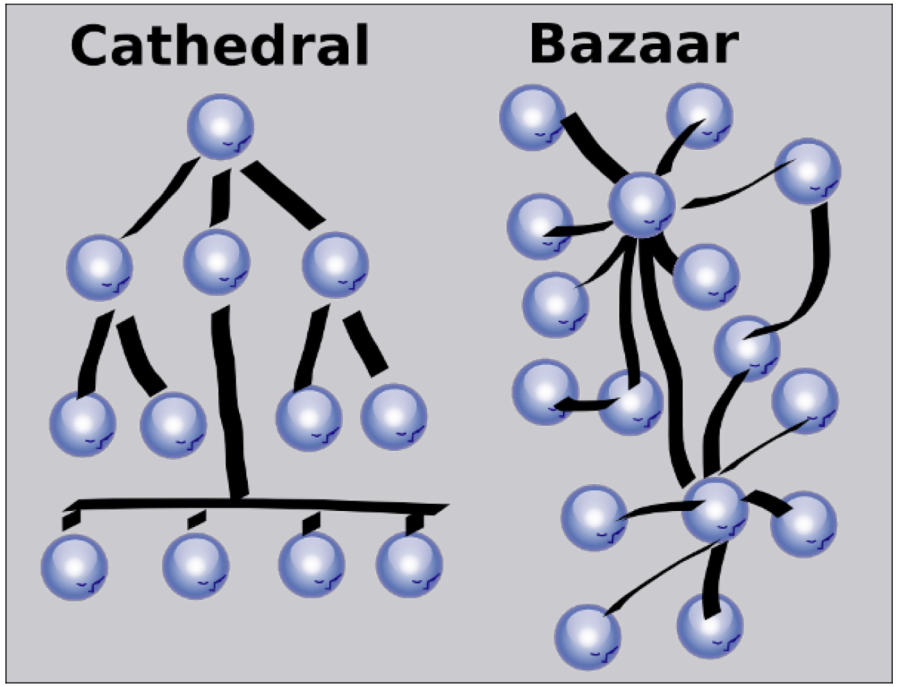
- Most readable structure
- Less likelihood of erroneously omitting function exit code
- Required by many safety standards.
  - IEC 61508
  - ISO 26262

Users demand Accountability

- Feature richness and completeness is not enough
- Adoption barrier unless there is a clearly identified entity that is responsible for the software and safety sign-off
- Main reason why adoption of open source software is limited for higher safety integrity levels
  - “Who is liable if something goes wrong?”
- Even with a certified offering, open or proprietary* and with a clearly accountable entity behind it, it is difficult to have early adopters (Nobody wants to be first).

How to approach certification in open-source

- Snapshotting a Source Tree (branch), validating it then controlling updates is a viable approach to software qualification
  - Build a cathedral on top of (or beside) the bazaar
- Getting supported feature set right is most important up front decision
  - The more you support, the more documentation and testing you are going to provide
- Automate as much of the information tracking as you can
- Auto-generate documents from test and issue tracking systems
- Get proof of concept approval from a certification authority as early as possible



The Ideal Project

- Has a split development model:
  - **Flexible open instance**: developed as usual in the open with community participation
  - **Auditable and controlled instance**: Branch with well defined scope developed with stricter rules and with an entity behind it.
- Auditable instance aligns with the open instance at a cadence dictated by necessity and certification cost.
- The entity running the auditable code base has experience with assessment and certification and sas ideally already been down this route before and has ideally gotten the blessing of users by way of product deployments
- An open source community helps enrich the open instance at a suitable pace by open collaboration. Everyone benefits from this instance.
- The owning entity maintains the auditable instance and takes on the certification qualification overheads. Users who want assurances engage with the owning entity (they get to point the finger).

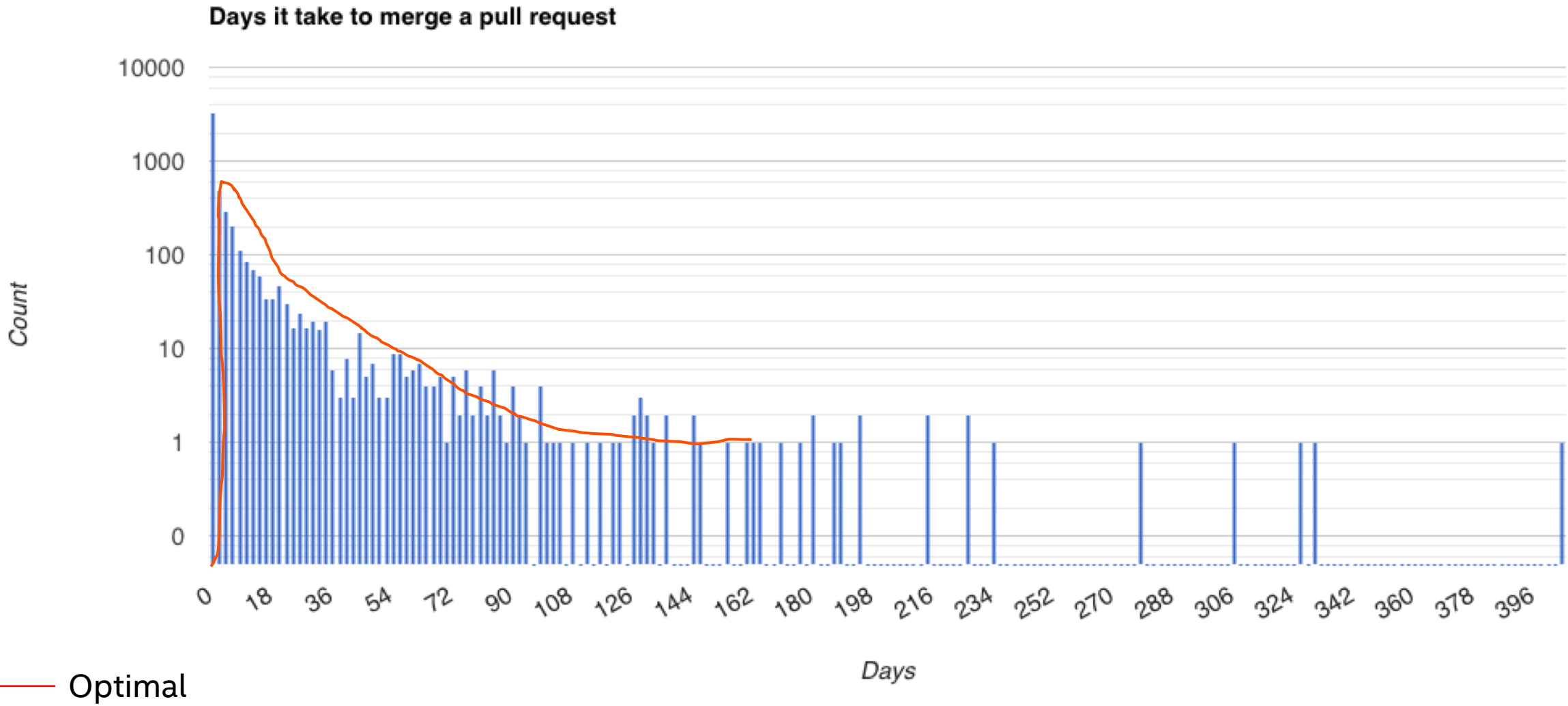
Example: Regulating the Bazaar

- Code is available publicly and can be scrutinized by anyone.
- Code Reviews and direct user feedback help improve quality

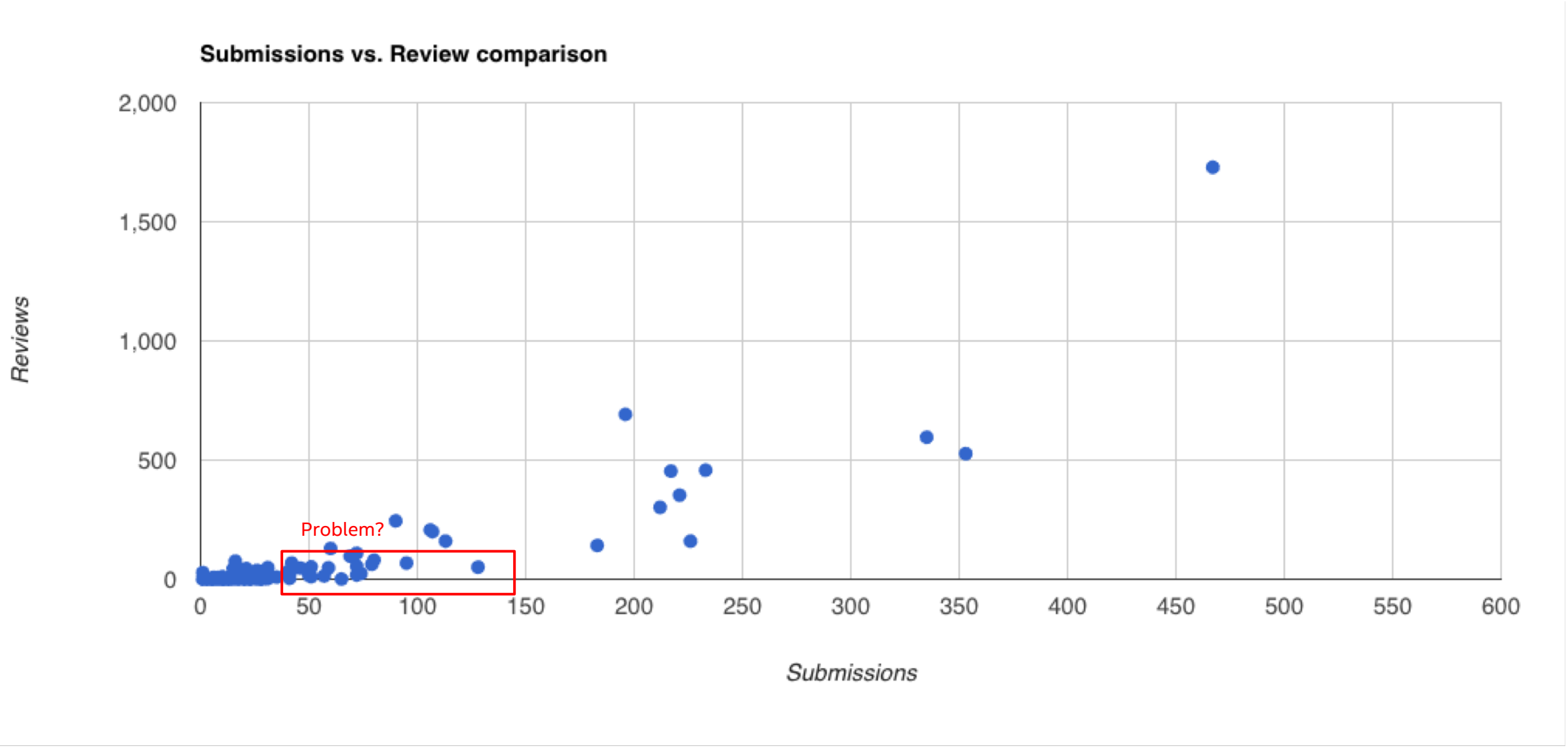
However...

- Do we have the right set of reviewers?
- Who gets to have the final say?
- How do we guarantee that the reviewer is aware of Safety implications?
- For how long should changes be reviewed?

Zephyr: Pull Request Processing Times



Contributions vs Reviews




Reviewers, Reviewers, Reviewers, Reviewers, ...

SafeRTOS did something similar, not quite!

- FreeRTOS-compatible alternatives from Wittenstein
- SafeRTOS was rebuilt from the same code base for compatibility.
- SafeRTOS has been rewritten and meets the requirements of the IEC 61508 safety standard.

Not the ideal model for open-source


Where are we with Zephyr?

… but we have the ingredients to get there fast.

Zephyr: a modular RTOS

Challenge

Many companies and business groups paying for different real time OS solutions, for small connected devices and embedded controllers.

This lead to costly, time consuming and divergent solutions for Intel and our customers

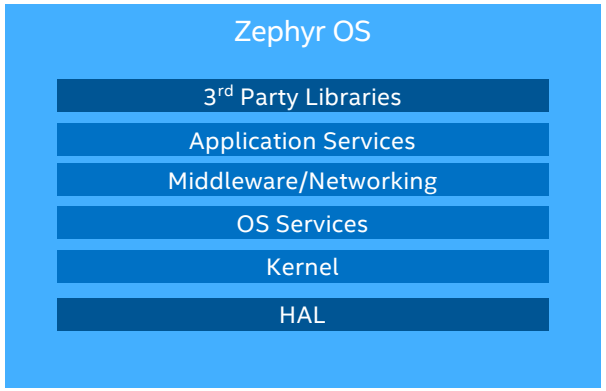
Solution, Zephyr

- A small, modular, open source, real-time operating system (RTOS) for use on connected resource-constrained and embedded controllers
- Supports diverse use cases and architectures
- Focused on safety, security, connections with Bluetooth support, and a full native networking stack
- Apache 2.0 license, hosted at Linux Foundation

Ecosystem Support

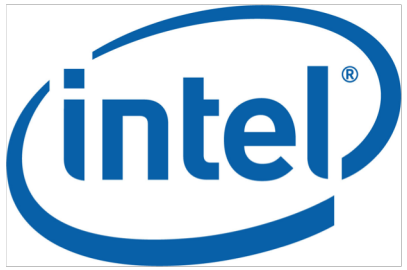
intel
ARC Synopsys
tensilica
RISC-V
ARM
Nios II Processor

Stack



Project Members

Platinum Members



Silver Members

Zephyr – A fully featured RTOS

Zephyr is a small, modular, open-source real-time operating system (RTOS) for use on resource-constrained systems covering diverse use cases and supporting multiple architectures.

Safety

- Thread Isolation
- Stack Protection (HW/SW)
- Quality Managed (QM)
- Build time configuration
- No dynamic memory allocation
- FuSA (2019)

Security

- User-space support
- Crypto Support
- Software Updates

Configurable & Modular

- Zephyr Kernel can be configured to run in as little as 8k RAM
- Enables application code to scale
- Configurable and Modular

Cross Platform

- Support for multiple architectures
- Native Port
- Developed on Linux, Windows and MacOS

Open Source

- Licensed under Apache II License
- Managed by the Linux Foundation*
- Transparent development
- Fork it on Github!

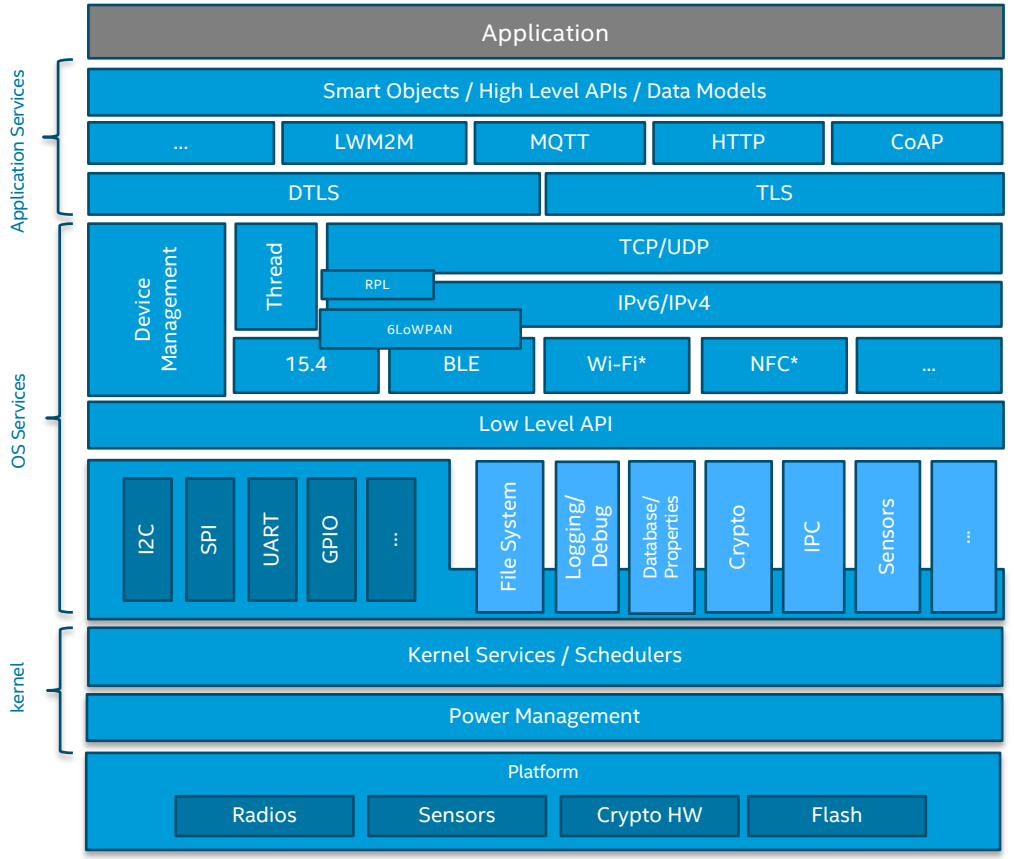
Connected

- Full Bluetooth 5.0 Support
- Bluetooth Controller
- BLE Mesh
- Thread Support
- Full featured native networking stack
- DFU (IP+BLE)

Zephyr is not an ingredient, Zephyr provides a complete solution.

Architecture and Key Features

- Highly Configurable, Highly Modular
- Cooperative and Pre-emptive Threading
- Memory and Resources are typically statically allocated
- Integrated device driver interface
- Memory Protection: Stack overflow protection, Kernel object and device driver permission tracking, Thread isolation
- Bluetooth® Low Energy (BLE 4.2, 5.0) with both controller and host, BLE Mesh
- Native, Fully featured and optimized networking stack
- Industrial Protocols



Fully featured OS allows developers to focus on the application

Why Zephyr?

The Zephyr OS addresses broad set of embedded use cases across a broad set of platforms and architectures using a modular and configurable infrastructure.

Address Fragmentation

- No single RTOS addresses broad set of embedded use cases across a broad set of platforms and architectures
- Disjoint use cases have led to fragmentation in RTOS space
- Existing commercial solutions force roll your own solutions and duplication of software components

Modular Infrastructure

- **Modular and configurable infrastructure** allows creation of highly compact and optimal solutions for different products from a **common** origin
- **Reuse a**llows NRE costs to be amortized across multiple products and solutions
- **Multi-architecture** support reduces platform switching costs and vendor lock-in concerns

Open-Source

- Roll your own is expensive & difficult to develop & maintain
- Permissively licensed corresponds to ease of adoption
- Corporate sponsorship assures long term commitment and longevity
- Community innovation has proven faster for progression and project development is a collaboration of industry experts

Feature Richness

- Need for a solution or semi-complete solution rather than just an ingredient.
- Lowers entry level barrier for new products and speeds up software delivery using existing feature and hardware support
- Encourages adherence to standards and promotes collaboration on complex features inside the organization
- Developers focus on the end-user facing interfaces instead of re-inventing low level interfaces

Reduce costs and improve efficiency through reuse

Zephyr Roadmap 2018/2019

- Safety and Security
  - FuSa Capable: Secure and harden the Kernel to meet IEC61508 SIL 3 (2019+)
  - Thread Isolation, User-space, Stack Protection
  - Development model and process with security and safety in mind
  - Secure and harden the Kernel (1.14)
  - MISRA-C 2012 Compliance (1.14)
  - Trusted Execution Environments (1.14)
- Expand use cases and application areas
  - Industrial, safety and security features (1.14)
  - Deep Embedded usages
  - Advanced Configurations and use cases: Multicore, SMP, AMP, .. (1.12)
- Introduce and support Zephyr as an E2E platform:
  - Bootloader (1.11)
  - Device Firmware Updates (1.11)
  - Cloud Connectivity
  - Development Tools
- Eco System, Portability
  - Improve support on Mac* and Windows* (1.11)
  - IDE integration (1.14)
  - 3rd Party Tools: Tracing, Profiling, Debugging... (1.13)
  - LLVM, Commercial compilers, .. (1.14)
  - Standard APIs and Portability: POSIX Layer (PSE54), BSD Socket (1.12), CMSIS RTOS v1 (1.13)

Roadmap to FuSA & Security Pre-Cert.

1. Limit the Scope

- Limit to officially supported and maintained code
- Start of the lowest layers and go up the stack

2. Robustness and operational safety

- MMU and MPU support
- Thread Isolation
- Stack Protection

3. Enhance and Increase Test Coverage

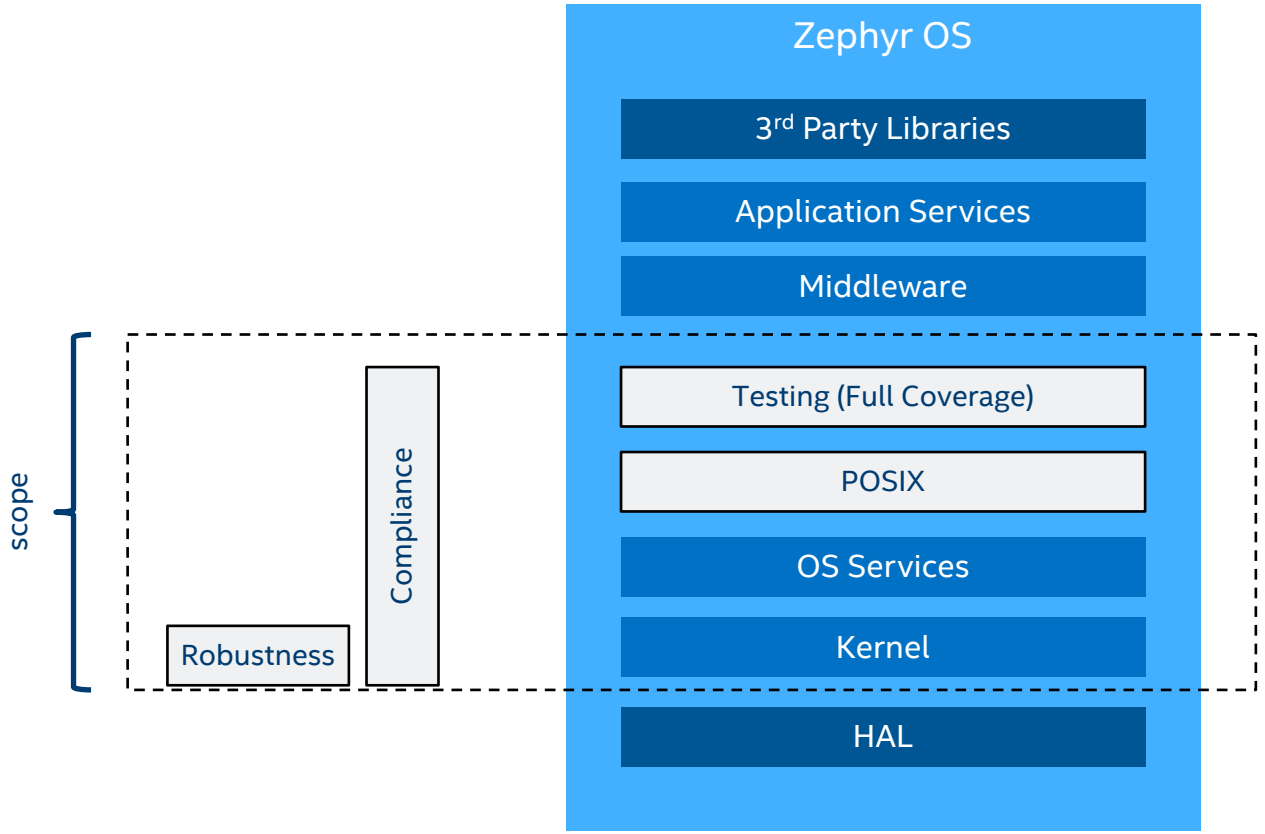
4. Compliance with coding and style guidelines, development process

- MISRA-C Compliance (MISRA-C:2012)

5. Well defined and Stable APIs

6. Portability

- Support POSIX APIs (PSE52, long term PSE54)



Candidate Standards

Coding for Safety, Security, Portability and Reliability in Embedded Systems:

- MISRA C:2012, with Amendment 1, following MISRA C Compliance:2016 guidance

Safety

- IEC61508: 2010 (SIL 3, but possibly SIL 4)
  - broadest for robotics and autonomous vehicle engineering companies. Reference for other standards in Robotics domain.
  - Sampled Certifications derived from IEC61508: Medical: IEC 62304; Auto: ISO 26262; Railway: EN 50128

Security

- Common Criteria (EAL4 but possibly higher levels EAL5,6)

Others

- Medical: FDA 510(K), ISO 14971, IEC 60601; Industrial: UL 1998, ??

Zephyr Long Term Support (LTS)

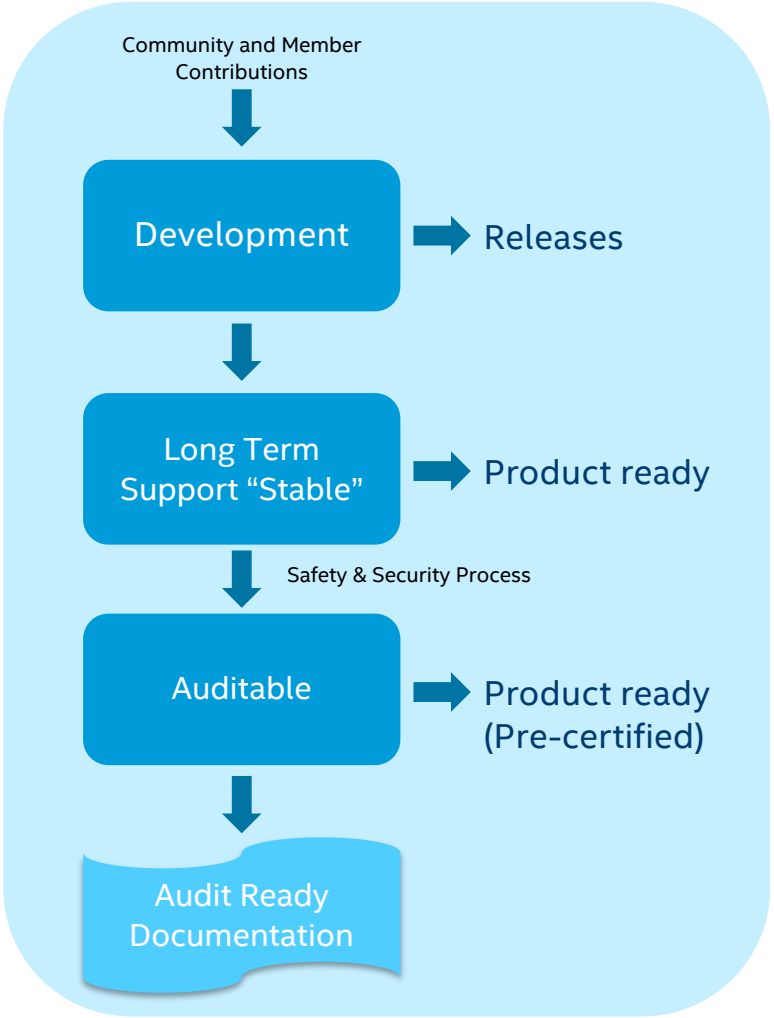
It is

- **Product Focused**
- **Compatible with New Hardware**: We will make point releases throughout the development cycle to provide functional support for new hardware.
- **More Tested**: Shorten the development window and extend the Beta cycle to allow for more testing and bug fixing
- **Certifiable**: The base for the auditable branch

It is not

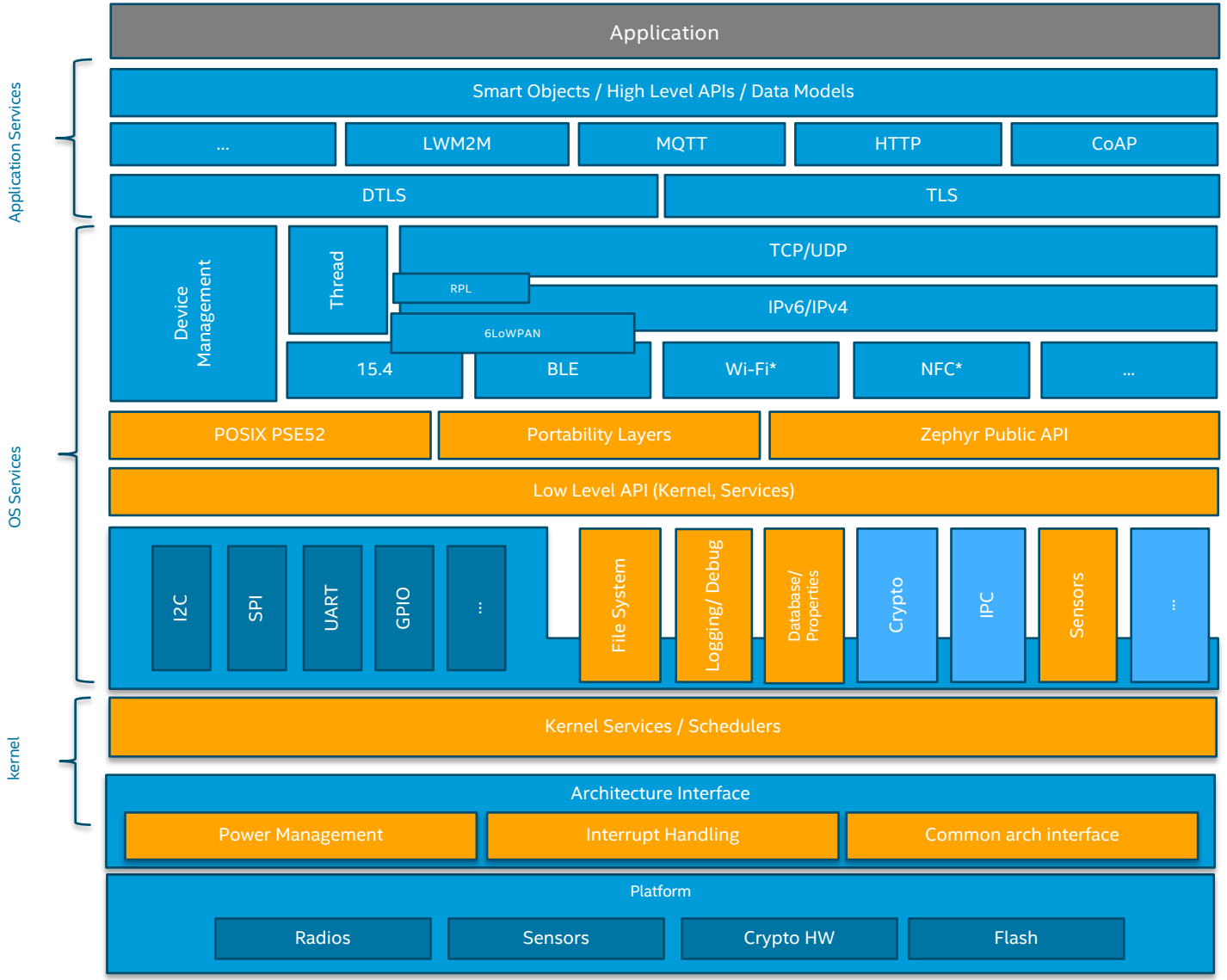
- **A Feature-Based Release**: focus on hardening functionality of existing features, versus introducing new ones.
- **Cutting Edge**

Auditable Code Base



- An auditable code base will be established from a subset of Zephyr OS features.
- Both code bases will be kept in sync from that point forward, but more rigorous processes (necessary for certification) will be applied before new features move into the auditable code base.
- Initial and subsequent certification targets to be decided by Zephyr project governing board.
- Processes to achieve selected certification to be determined by Security Working Group and coordinated with the TSC.

Scope for FuSA (in orange)



Summary

- Functional Safety and Security requirements need to coexist with the open-source nature of the project
- Quality needs to be driven on the project level
  - Need to showcase our quality process and test plans publicly
  - Drive adoption through quality managed release process
- Manage Developer and Contributor Expectations
- Continue innovating on main tree while hardening and stabilizing Zephyr LTS, ,the base for any auditable branches
- Need to officially establish accountability and trusted “entity”, i.e. with Certification Architect role in the project

Get Started

Resource	Pointer
Website	http://www.zephyrproject.org/
Documentation	http://docs.zephyrproject.org/
Git Repository (Code)	https://github.com/zephyrproject-rtos/zephyr
Issues	https://github.com/zephyrproject-rtos/zephyr/issues
Mailing lists	https://lists.zephyrproject.org/mailman/listinfo

Q/A