LINUX SAFETY VERIFICATION

A PROCESS FOR USING LINUX IN SAFETY-CRITICAL ENVIRONMENTS

Dr. Lukas Bulwahn | April 25th, 2018

BMW GROUP
THE NEXT 100 YEARS
BMW Car IT

MOTIVATION

BUSINESS MOTIVATION

THE WALL STREET JOURNAL.

ESSAY

Why Software Is Eating The World

By Marc Andreessen

August 20, 2011

Main message: Software innovations will disrupt every industry, even very established industries.

Various companies in the mechatronics industry are struggling with this change being driven by software innovations and software industry competitors.

Give profits to the software vendors or invest to explore alternatives?

THE HISTORY OF UNIX

1980s

Various UNIX operating systems in the market. Vendors largely controlling their users, with an incompatible vendor-lock-in mess.

1990s

Linux is born. Linux forms as the coalition of those tired users and losers.
With an open-source collaboration model, Linux builds a strong ecosystem of users and software companies.

Today

Linux (in 2017):

- 90% public cloud workload
- 62% embedded market share
- 99% supercomputer market share
- 82% world's smartphones

(Source: https://www.linuxfoundation.org/publications/2017-state-of-linux-kernel-development/)

Facebook, Google, IBM, Intel... have Linux kernel teams for their Linux operating system.
They are in control of the software chain & stack,
although they don't even sell software, but services or hardware.

HISTORY REPEATS ITSELF

Mechatronic industry is now at the same crossway
for safety-critical operating systems
to run complex algorithms and software.

How to create a healthy ecosystem of safety-critical operating systems
to focus on innovative software functions?

STRENGTHS OF THE LINUX OPERATING SYSTEM

The Linux kernel has:

- Large Development Ecosystem
- Security Capabilities
- Multi-Core Support
- Unmatched Hardware Support
- Many Linux Experts at all levels available

The Linux kernel is missing:

- Real-time Capabilities
- Proven Safety-compliant Development Process

Can these gaps be closed?

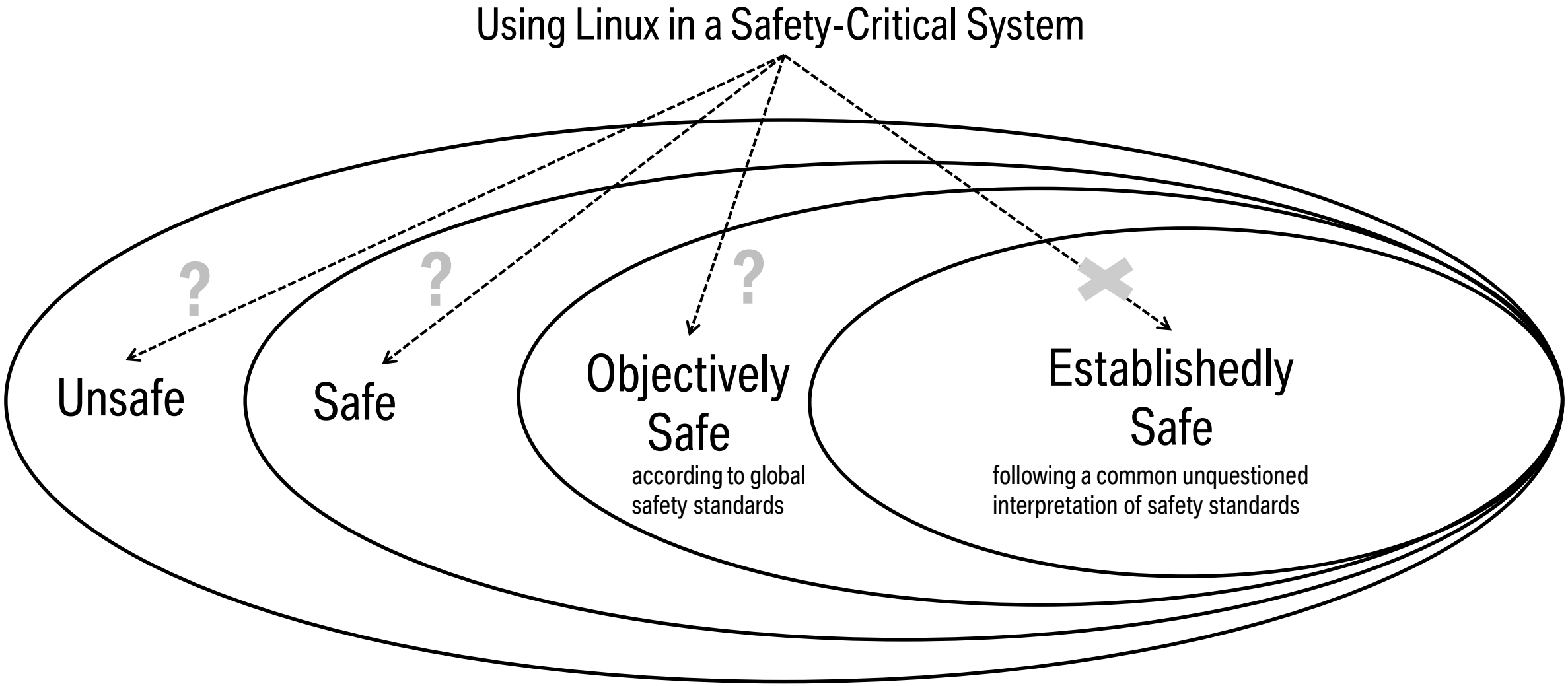
THE OSADL SIL2LINUXMP PROJECT

- **Mission:**
  - **Provide procedures and methods** to qualify Linux on a multi-core embedded platform at safety integrity level 2 (SIL2) according to IEC 61508 Ed 2.
  - **Show feasibility** of **procedures and methods** on a real-world system
  - **Show potential** for collaboration and re-use of Linux kernel analysis

- **Collaborative project of industrial & research partners**
  - project running since 2015, organized by OSADL
  - Full members: BMW Car IT, Intel (since ‘17), A&R Tech, KUKA, Sensor-Technik Wiedemann (full members till ’16, reviewing members in ‘17)
  - Reviewing members: Bosch, Elektrobit, Hitachi, Linutronix, MBDA Italia, MEN Mikro Elektronik, Mentor, OpenSynergy, Pilz GmbH & Co. KG, Renesas, Vienna Water Monitoring Solutions
  - Academic members: Alexey Khoroshilov (ISP RAS), Kinggo Chow (Lanzhou Univ.), Julia Lawall (Inria/LIP6), Frank Tränkle (HS Heilbronn)
  - Experts from certification bodies: Bernhard Nölte (TÜV Süd), Oliver Busa, Robert Heinen, Hendrik Schäbe (TÜV Rheinland)
  - SIL2LinuxMP core working team: Nicholas McGuire, Andreas Platschek, Lucas Böhm, Markus Kreidl (OpenTech)

GENERAL ASSUMPTIONS

THE LINUX SAFETY RESEARCH QUESTION



BALANCE OF SAFETY SYSTEM ENGINEERING

First Law of System Safety Engineering

A System is safe by a proper composition of system elements.

Corollary

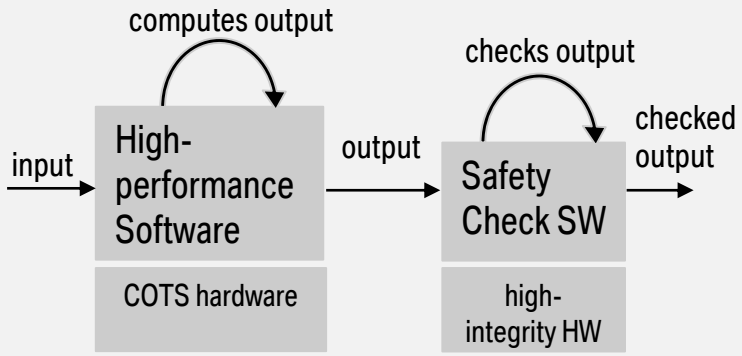
A System with Linux is safe if Linux is composed properly into the overall system.

Task: Find out how to compose Linux properly into a system

- Determine which safety capabilities can be argued for Linux?
- Find proper balance between assigned safety requirements and feasible capabilities
- Provide full evidences for required capabilities

POSSIBLE SYSTEM ARCHITECTURES

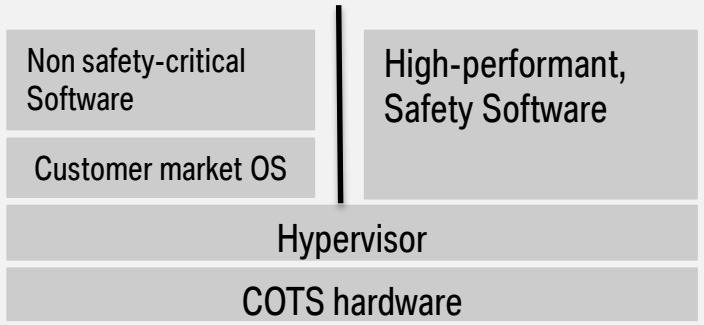
System Architecture A



Assumptions/Drawbacks:

- Needs decomposition of safety of complex function to a simple checking on lower performance high-integrity HW
- Checking must detect subtle errors from HW and OS of high-performance computation.

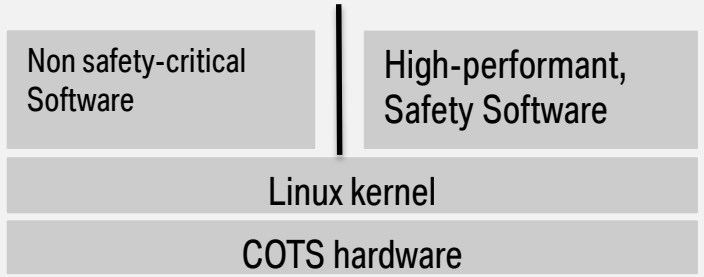
System Architecture B



Assumptions/Drawbacks:

- Hypervisor & HW guarantees isolation
- Safety SW without underlying OS
- If needed, scheduling, multi threading & file system is implemented in safety SW

System Architecture C



Assumptions/Drawbacks:

- Linux kernel provides sufficient isolation
- Safety SW uses Linux scheduling, multi threading, file system etc.
- Parts of Linux kernel and HW functionality must be qualified.

SIL2LinuxMP Project focusses on work for system architecture C.

NOTABLE FACTS ON THE LINUX KERNEL

- **Linux is a large software project with an impressive change rate**
  - 23 million lines of code & 2 million lines of documentation
  - 14,000 commits in every release/every 70 days
  - 17,000 developers in total history, 1,700 developers in each release
  - kernel developers are affiliated with many different companies or act as individuals
- **Development process**
  - Highly transparent due to public visible collaboration
  - Process defined and enforced by social contract, but not legal working contracts
- **Stabilizing phase of Linux LTS kernel versions**
  - 7360 bug-fix commits in the 4.9 branch until v4.9.95 => ~100 bugs corrected each week
  - detected by continuous developer review, various verification activities and execution on billion devices

RESEARCH RESULTS

OVERVIEW OF RESULTS

- Qualification Route
- Systematic Approach to Replace Methods
- **Hazard-driven Decomposition, Design and Development**
- **Software Layers-of-Protection Analysis**
- SIL2LinuxMP Architecture
- **Statistic Modelling**
- **Verification Activities**
- **Safety-Impact Analysis**

HAZARD-DRIVEN DECOMPOSITION, DESIGN & DEVELOPMENT (HD³)

- **Goal** of the system analysis: **Precise technical safety requirements** on lower levels with traceability on system's safety impact

- **First naive system safety engineering**
  "The syscall open() is used in a safety-critical application and must work correctly."
  - Problem: Too imprecise to guide further testing, verification and validation activities

- **Safety engineering with Hazard-driven Decomposition, Design & Development (HD³)**
  - Dedicated method to achieve more precision on complex systems with multiple levels for fault avoidance & detection
  - Results in 12 constraints on syscall open()!
  - Specific testing and verifications under those specific conditions is now feasible

System Call: open()

Origin: I/O Setup (ST_3020-ST_3029), Create File (ST_3040-ST_3049)

HAZOP IDs: SD_3520-SD_3529

References: 1) man 2 open 2) POSIX 1003.13 summary in section 6.6.1.4 3) open()

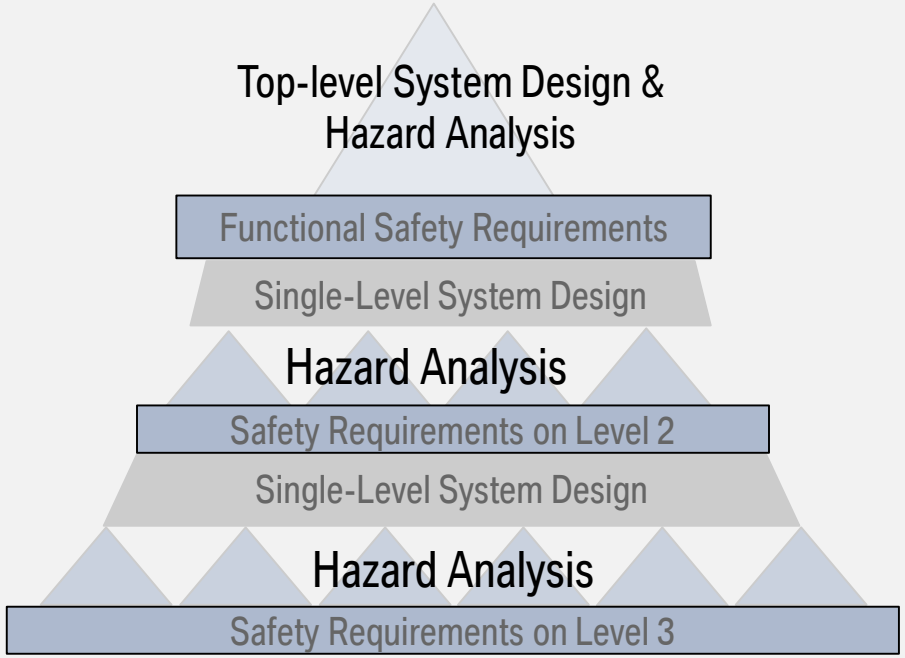
Related SACs: STS_3013 (minimize access priviledges), STS_3014 (access modes pre-determined), STS_3015 (init opens/creates files), STS_3019 (restricted usage of lseek()), STS_3028 (defined file creation flags), STS_3029 (check new files existence), STS_3030 (file creation logged), STS_3034 (file creation in non-RT), SDS_3508 (magic numbers prohibited), SDS_3509 (write files exclusively), SDS_3510 (set and verify permissions), SDS_3512 (file namespace specified), SDS_3513 (random bits in file names)

HAZARD-DRIVEN DECOMPOSITION, DESIGN & DEVELOPMENT

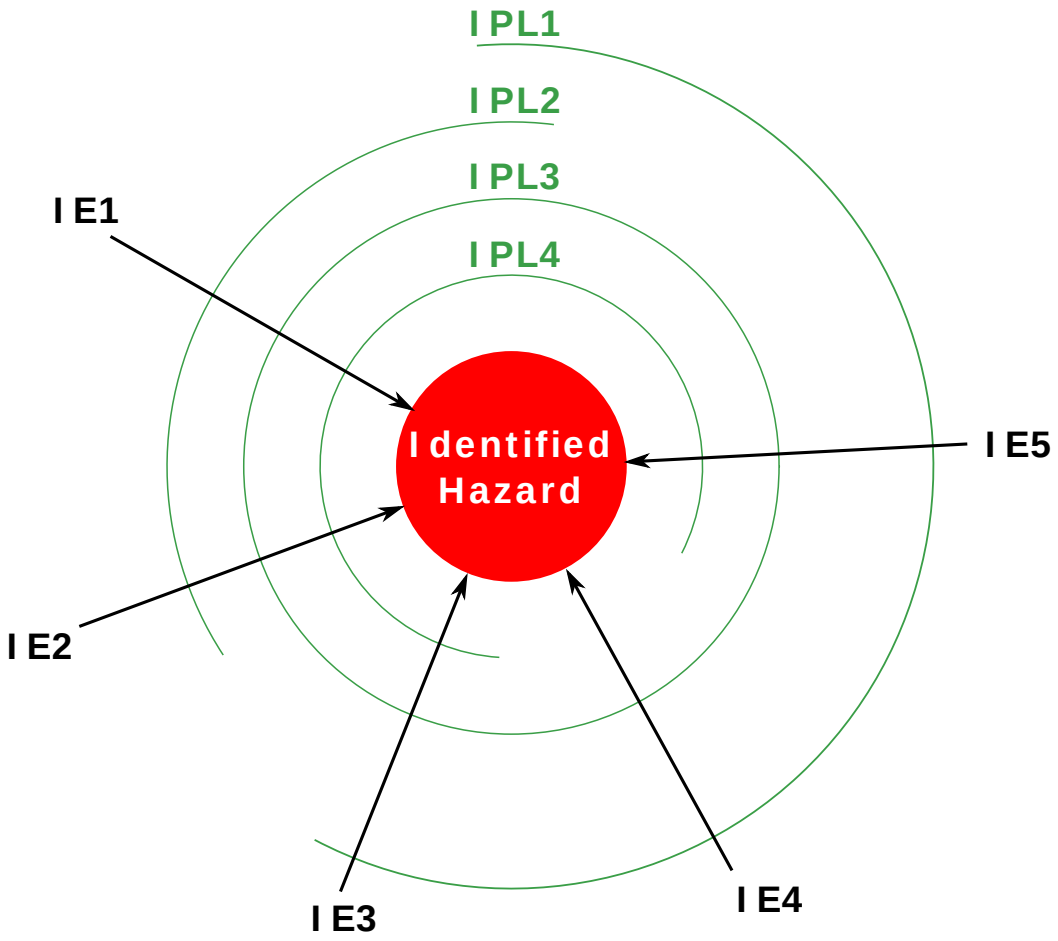
System Safety Engineering Method to obtain **precise** technical safety requirements on lower levels with **clear safety impact**

First Naive System Safety Approach

Hazard-driven Decomposition, Design & Development (HD³)

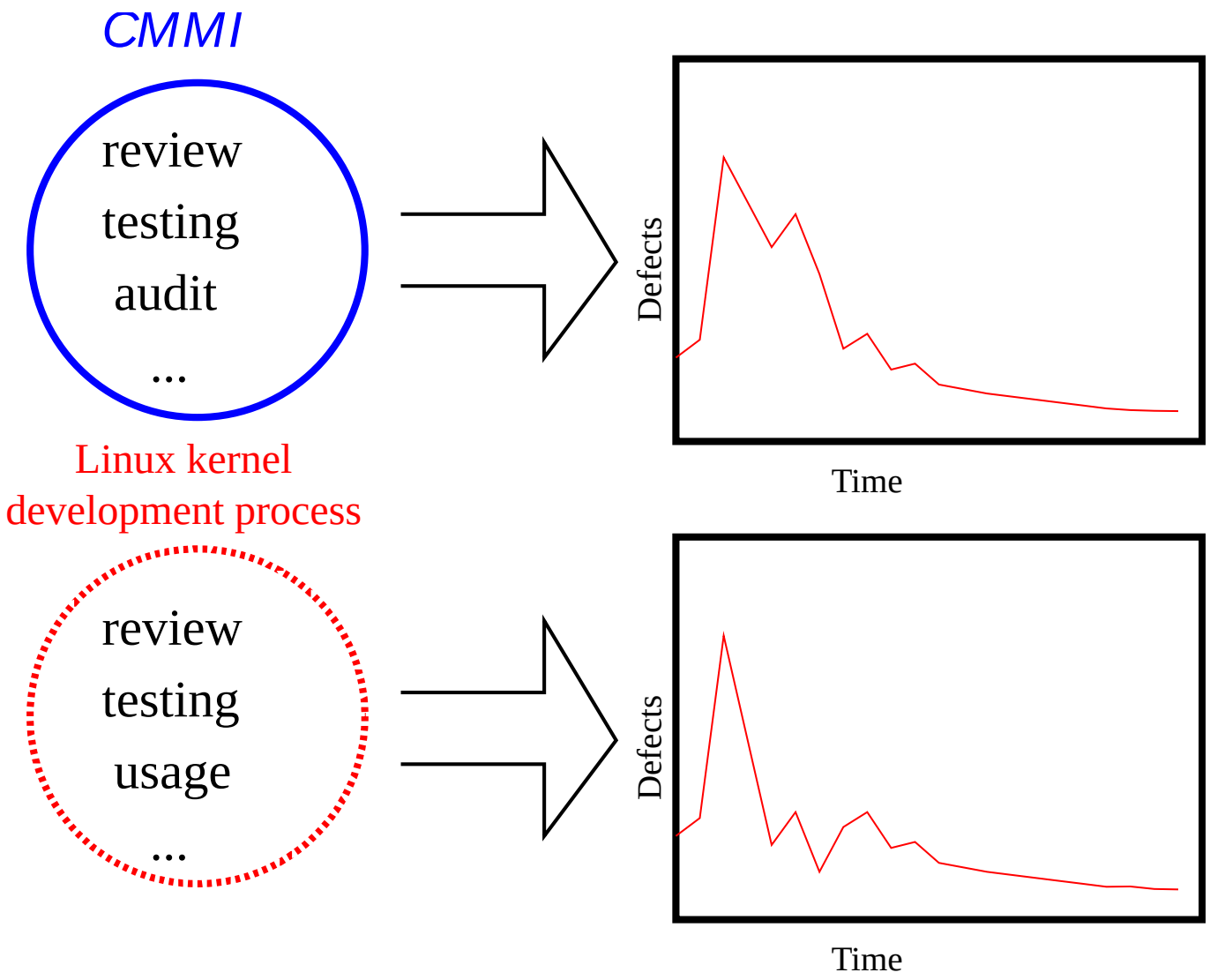


SOFTWARE LAYERS-OF-PROTECTION ANALYSIS



- Independent Protection Layers (IPL) are in Software (mostly security mechanisms).
- IPLs designed to provide isolation, but no quantification possible.
- Independence of Layers assured by analysis of:
  - software architecture and implementation
  - development meta data, e.g., development history and contributors.

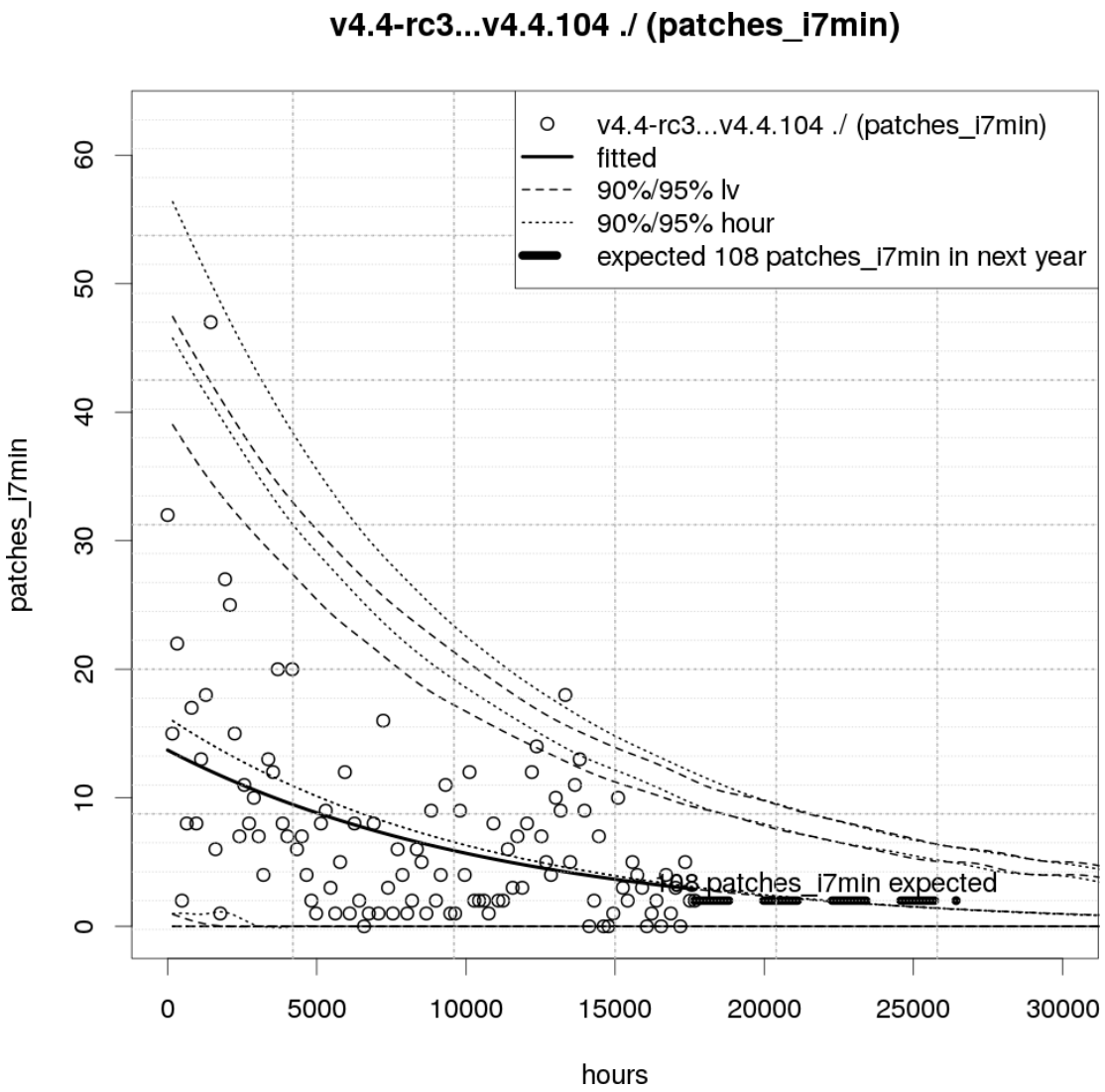
STATISTIC MODELLING (PRINCIPLE)



Idea:

Confirm process effectiveness qualitatively by statistic modelling of bug defect detection over time.

STATISTIC MODELLING (EXAMPLE)



Still highly preliminary and under discussion:

Needs review of experts in quantative software engineering and statisticians

In other domains, statistics is used intensively to prove facts empirically, but not yet so much in software development processes.

We need to ensure that we do not:
„incidentally over-interpret measured data, present data in not so useful ways, and/or draw unsupport conclusions from statistical analyses“
(Prof. Dr. Wolfgang Maurerer, Siemens/OTH Regensburg)

NEEDED QUALITY ASSURANCE ACTIVITIES FOR SAFETY-CRITICAL SYSTEMS

- **Ongoing and future quality assurance activities**
  - **Coding style**
    - Respect existing coding style
    - Provide evidences for its quality
    - Monitor and motivate its compliance
  - **Testing**
    - Extend tests of the Linux Test Project for the determined syscalls
  - **Static analysis**
    - Detect more bug patterns and bug classes with coccinelle, sparse et al.
  - **Change management**
    - Track if bug fixes from main line are consequently backported
    - Analyze which kernel bugs & fixes impact the systems' safety
- Activities focus on parts of the Linux kernel relevant for the systems' safety

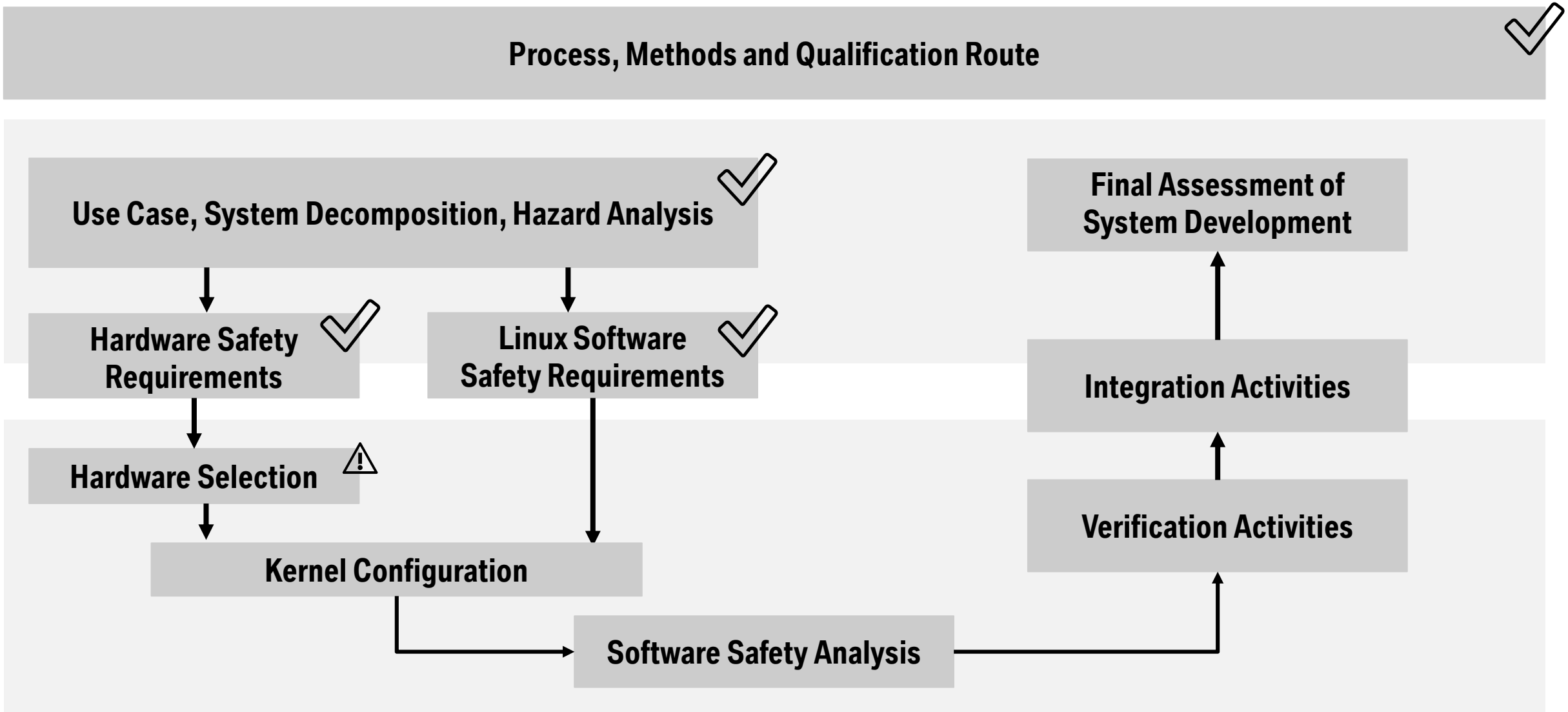
MAINTENANCE ACTIVITY: SAFETY IMPACT ANALYSIS

- **Rationale** for doing safety impact cause analysis during operations
  - Safety standard requires continuous monitoring and analysis of identified issues
- **Implementation** with the Linux kernel development
  - Bugs are continuously found in Linux
  - Bug-fix commits are backported to the affected LTS branches
  - Then all product developers must determine if the bug and bug fix impact systems' safety of each system

- **Process** for each bug-fix:
  - **Step 1** (once for each bug-fix):
    - **Kernel analysis team** describes the impact of a kernel bug on user space and its bug-fix in high detail
    - Analysis independent of the specific system, **one collaborative team**
  - **Step 2** (for each bug-fix and each system):
    - **System analysis team** judges if the described bug-fix impacts the system and the system's safety.
    - One team for each system employing Linux

CONCLUSION

CURRENT USE CASE STATUS



NEXT STEPS AND OPEN CHALLENGES

- Find Suitable Hardware with an appropriate Safety Manual
  - e.g., with guarantees for suitable systematic capabilities on the memory management unit (MMU).
- Increase Confidence in Statistic Modelling
  - Consult with experts in quantative software engineering and statisticians
- Define Maintenance Process
- Implement Verification Activities
- Tie Parts together in the project‘s Use Case
- Grow Circle of Collaborative Companies to continue investigation and shared maintenance

CONCLUSION

- To learn more, **join the SIL2LinuxMP Safety-Critical Linux working group**
- Contact Nicholas McGuire at safety-at-osadl.org or contact me at lukas.bulwahn-at-bmw.de

- How you will learn more?
  - Quarterly 3-day Workshops of project participants (software and safety engineers)
    - Two more Workshops planned for 2018
    - First Participation is free of charge and commitments
  - Various documents of mentioned results are now available and under review of participating companies.

- **Using Linux** in safety-critical systems is **feasible**.
- But still much to learn, understand, interpret and do.

COPYRIGHT ON THIS PRESENTATION