RUST DORTMUND

2025-09-03

RUST IN AUTOMOTIVE

BY CHRISTOF PETIG

DEVELOPING SAFETY-CRITICAL APPLICATIONS EFFICIENTLY AND QUALITATIVELY

About me

1983: Assembly, Basic

1989: C

1996: C++

2004: Automotive

2020: Rust

SAE

AUTOSAR

WASI 0.3

veloren.net

Two adult kids

Safety Standards

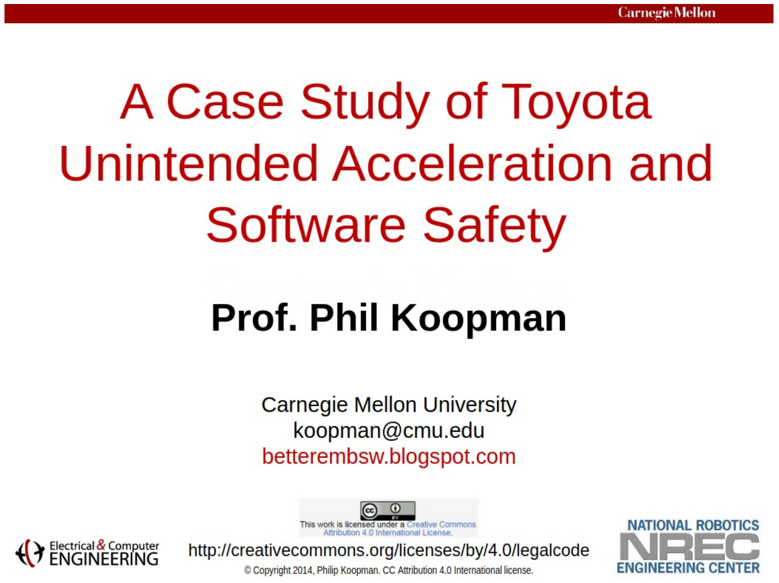
- IEEE IEC 61508
- ISO 26262
- RTCA DO-178C
- MISRA
- CERT

Very long documents with many details!

Why should we care?

Why safety?

… to prove in court that a fatality isn't your fault



Carnegie Mellon

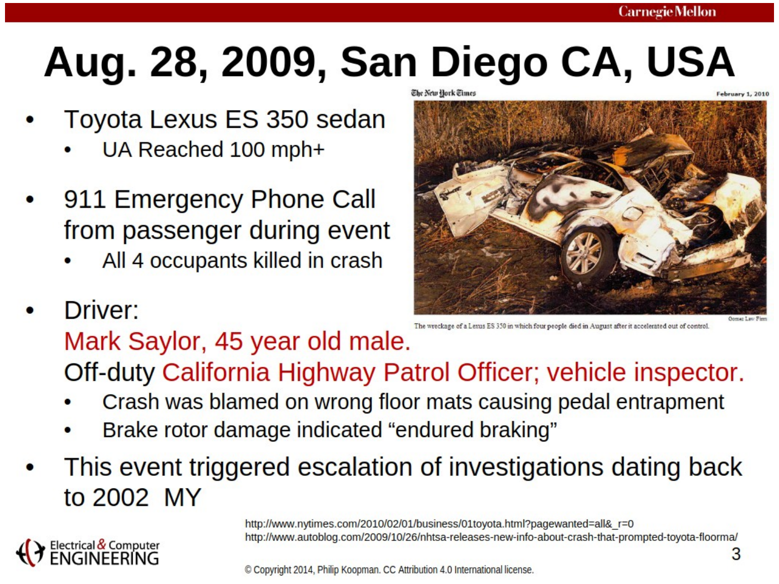
A Case Study of Toyota Unintended Acceleration and Software Safety

Prof. Phil Koopman

Carnegie Mellon University
koopman@cmu.edu
betterembsw.blogspot.com

Electrical & Computer ENGINEERING

NATIONAL ROBOTICS NREC ENGINEERING CENTER



Carnegie Mellon

Aug. 28, 2009, San Diego CA, USA

- Toyota Lexus ES 350 sedan
  - UA Reached 100 mph+
- 911 Emergency Phone Call from passenger during event
  - All 4 occupants killed in crash
- Driver:
  Mark Saylor, 45 year old male.
  Off-duty California Highway Patrol Officer; vehicle inspector.
  - Crash was blamed on wrong floor mats causing pedal entrapment
  - Brake rotor damage indicated "endured braking"
- This event triggered escalation of investigations dating back to 2002 MY

The New York Times   February 1, 2010

Gomez Law Firm

The wreckage of a Lexus ES 350 in which four people died in August after it accelerated out of control.

http://www.nytimes.com/2010/02/01/business/01toyota.html?pagewanted=all&_r=0
http://www.autoblog.com/2009/10/26/nhtsa-releases-new-info-about-crash-that-prompted-toyota-floorma/

Electrical & Computer ENGINEERING

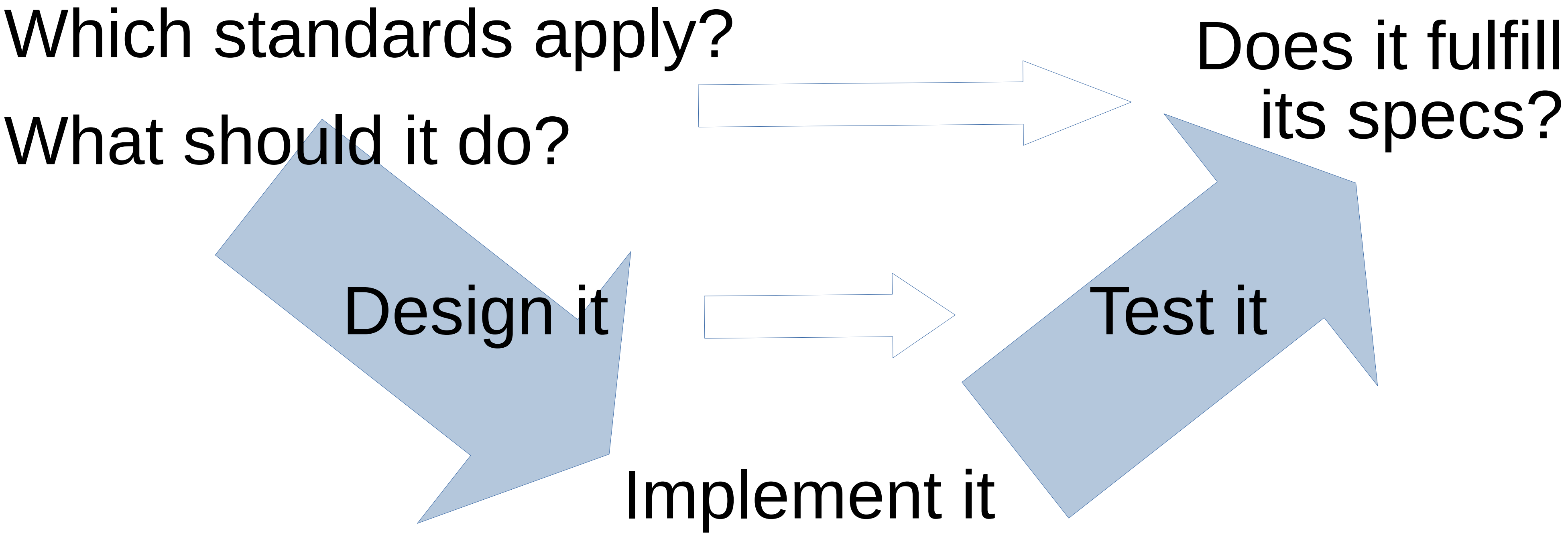
YouTube
DKHa7rxkvK8

https://users.ece.cmu.edu/~koopman/lectures/

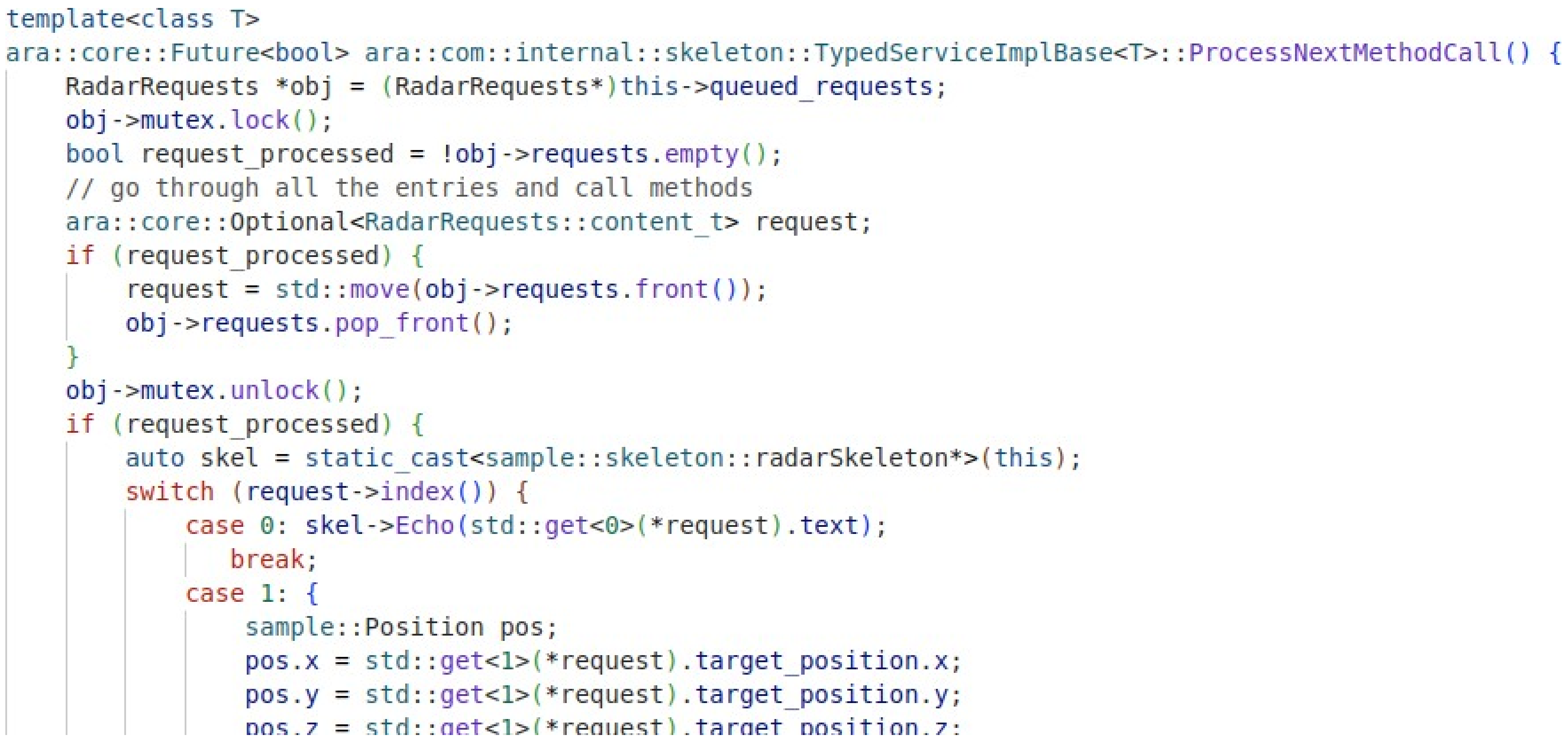
Possible root causes

- Memory corruption
- Stack overflow
- Many global variables
- Runtime tasks exceed slots
- Cosmic radiation

V-Model


Which standards apply?
What should it do?
Design it
Implement it
Test it
Does it fulfill its specs?

Is this code correct?



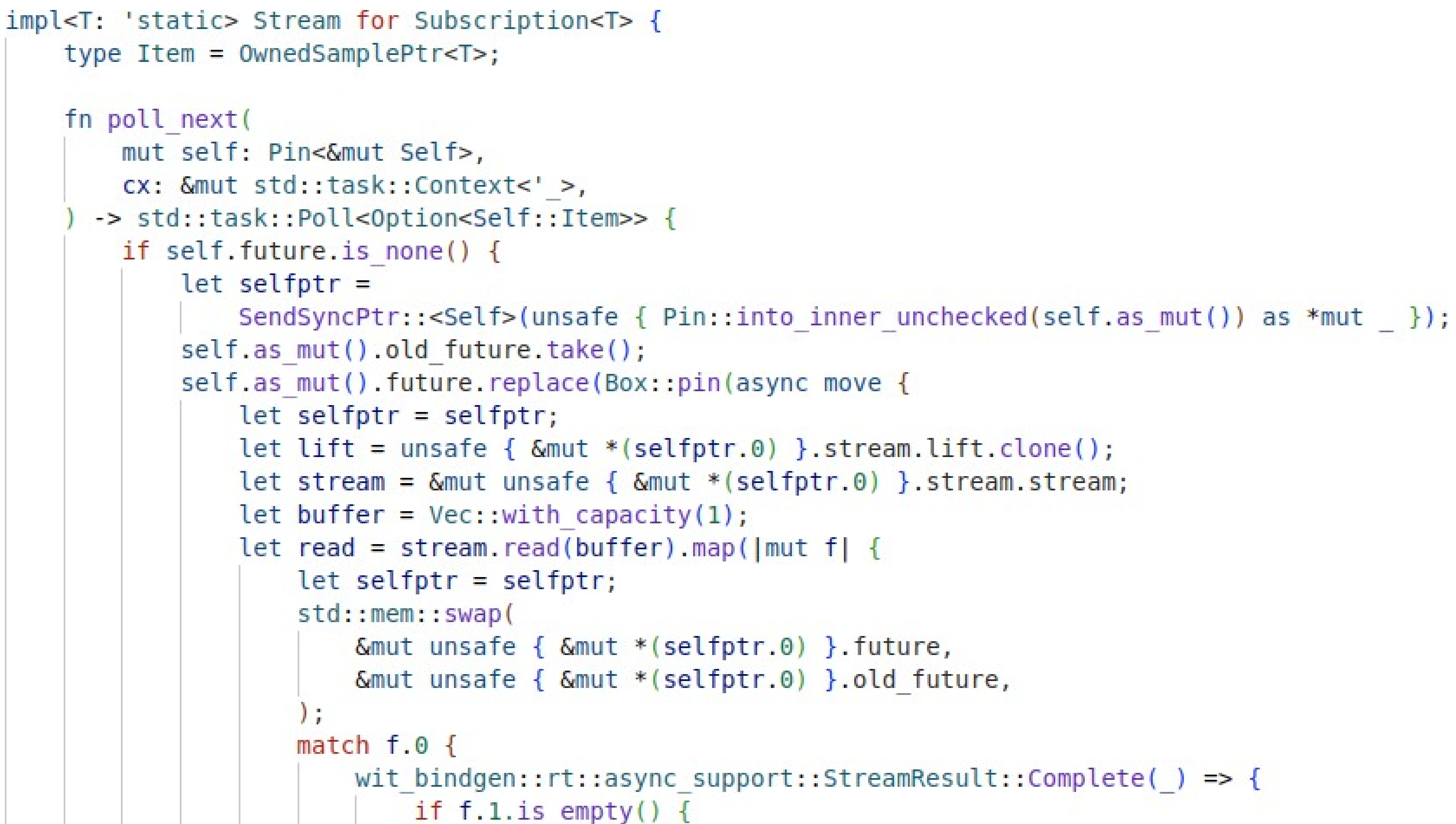
```
template<class T>
ara::core::Future<bool> ara::com::internal::skeleton::TypedServiceImplBase<T>::ProcessNextMethodCall() {
    RadarRequests *obj = (RadarRequests*)this->queued_requests;
    obj->mutex.lock();
    bool request_processed = !obj->requests.empty();
    // go through all the entries and call methods
    ara::core::Optional<RadarRequests::content_t> request;
    if (request_processed) {
        request = std::move(obj->requests.front());
        obj->requests.pop_front();
    }
    obj->mutex.unlock();
    if (request_processed) {
        auto skel = static_cast<sample::skeleton::radarSkeleton*>(this);
        switch (request->index()) {
            case 0: skel->Echo(std::get<0>(*request).text);
                break;
            case 1: {
                sample::Position pos;
                pos.x = std::get<1>(*request).target_position.x;
                pos.y = std::get<1>(*request).target_position.y;
                pos.z = std::get<1>(*request).target_position.z;
```

Why Rust?

- Prevent errors at compile time
- Global enforcement of a single write exclusive or multiple reads

- Excellent tooling
- Fun

Is this code correct?



```rust
impl<T: 'static> Stream for Subscription<T> {
    type Item = OwnedSamplePtr<T>;

    fn poll_next(
        mut self: Pin<&mut Self>,
        cx: &mut std::task::Context<'_>,
    ) -> std::task::Poll<Option<Self::Item>> {
        if self.future.is_none() {
            let selfptr =
                SendSyncPtr::<Self>(unsafe { Pin::into_inner_unchecked(self.as_mut()) as *mut _ });
            self.as_mut().old_future.take();
            self.as_mut().future.replace(Box::pin(async move {
                let selfptr = selfptr;
                let lift = unsafe { &mut *(selfptr.0) }.stream.lift.clone();
                let stream = &mut unsafe { &mut *(selfptr.0) }.stream.stream;
                let buffer = Vec::with_capacity(1);
                let read = stream.read(buffer).map(|mut f| {
                    let selfptr = selfptr;
                    std::mem::swap(
                        &mut unsafe { &mut *(selfptr.0) }.future,
                        &mut unsafe { &mut *(selfptr.0) }.old_future,
                    );
                    match f.0 {
                        wit_bindgen::rt::async_support::StreamResult::Complete(_) => {
                            if f.1.is_empty() {
```

Confidence in correctness

Lars Bergstrom, Google, Rust nation UK 2024

"… of this survey … so in comparison to code in other languages, **how confident** do you feel that your team's Rust code is **correct**?

Answer: **85 percent** of the people.

That is a massive number ... Eighty-five percent of people believe that their Rust code is more likely to be correct than the other code within their system."

Rust for safety

- ANSSI guidelines
- SAE JA1020 (coming soon)
- Safety Critical Rust Consortium

- Compiler qualification ferrocene, ADAcore, HighTec
- Crate qualification tbd

Efficiency & Quality

Lars Bergstrom, Google,
Rust nation UK 2024

"In every case we've seen a **decrease** by more than **2x** in the amount of **effort** required to both build the services in Rust as well as maintain and update those services written in Rust"

My personal take is **another 2x** in efficiency gain once functional **safety** rules are added to the project
(due to MISRA, safety guidelines, code checkers, etc.)

Even Hobby projects, like Veloren, **enforce** clippy rules for Rust on each commit.

Rust in automotive

Public information

- Toyota: Rust Foundation member
- Ampere: Rust trainings
- Volvo: Rust safety presentations
- VW: SommR

- VDA: s-core
- AUTOSAR Vector, ETAS, Elektrobit, HighTec

https://eclipse-score.github.io/

We support both C++ and Rust programming languages. Software development is done in both languages. Decision which language to choose is done during architecture process.

In general, C++ should be used **only for the existing** modules, that are taken over or referenced by the SCORE project.

For new features and components we aim to develop the code mostly in Rust, as it seems to be **more suitable** for development compliant to ISO 26262:2018.

Safety Culture

Toyota commemorates the 911 call each year.

Safety is an ongoing process involving everyone.