Towards Verified and Certifiable Subsystems

Burkhart Wolff

Univ. Paris-Saclay / IRT SystemX: Project PST

http://www.lri.fr/˜wolff

Abstract

Title: Towards Verified and Certifiable Subsystems

Abstract: This talk addresses the problem of comprehensive verification of (safety-critical) subsystems including processor, OS, and application function. Modeling, Refinement-Proofs, Code - and Document Generation were done in Isabelle/HOL. Particular emphasis is done on the aspect of document-generation targeting a formal certification process; the approach is centered around a central document from which all artefacts were generated in oder to ensure their coherence both in formal as well semi-formal aspects.
The approach is demonstrated for the Odometric Function of a railway system implemented on top of seL4 and a SabreLight Board. The toolchain build around Isabelle is called CVCE.

Overview

- The Case Study:
  An Odometric Subsystem
- Verification Methodology
- Certification Methodology
- Scaling up:
  Integrating the Odometer Business Logic into the seL4-OS stack

An Odometer

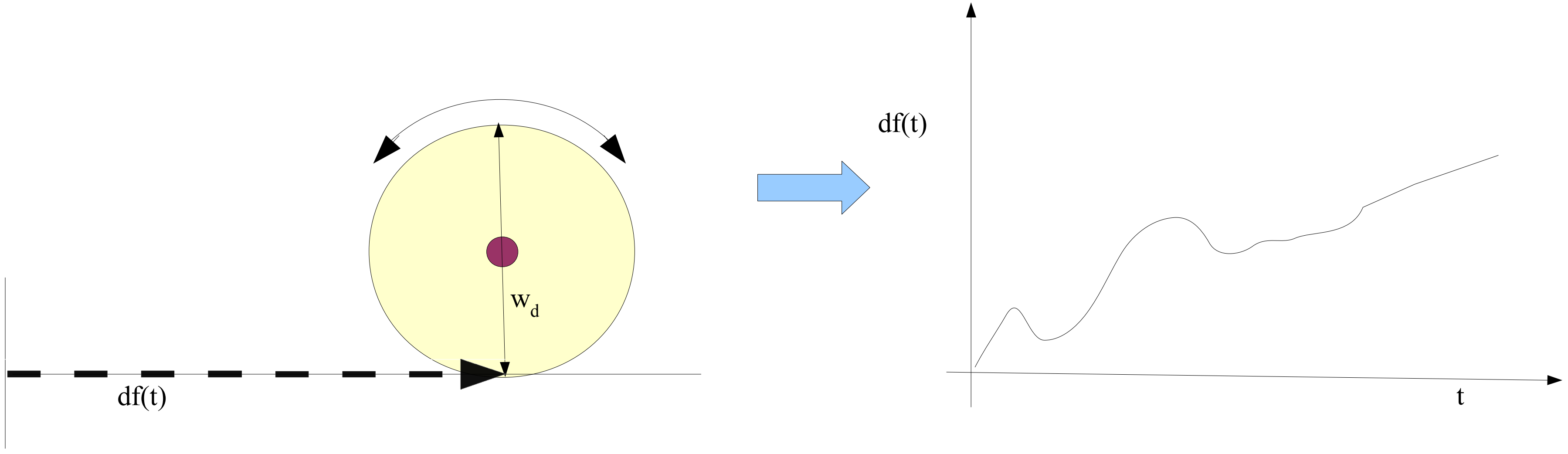
- Train position and movement detec- tion system
- Makes the decision that a train comes to a halt
- Hard- and software: embedded system
- key safety critical component

An Odometer

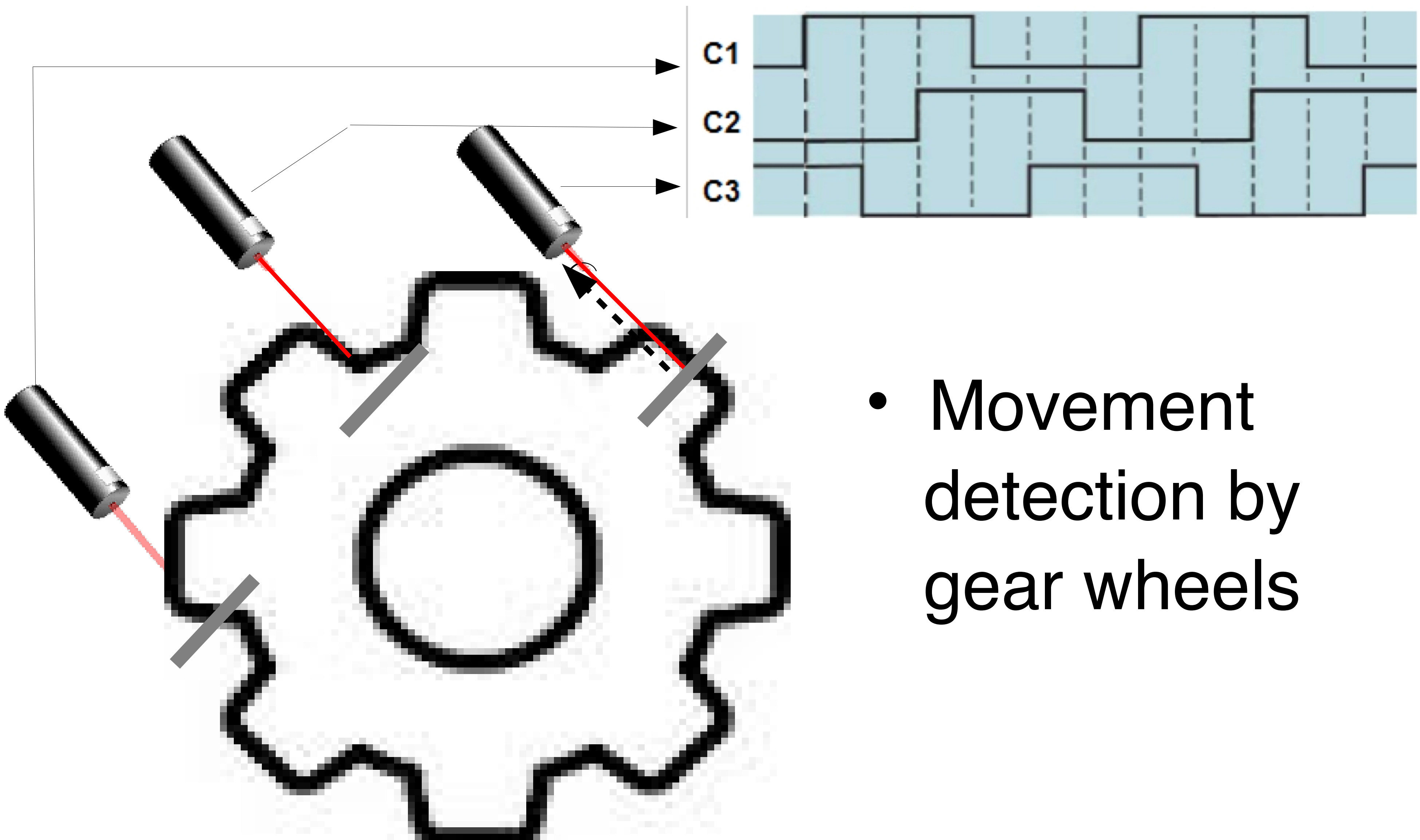
- Train position and movement detec-tion system
- Makes the decision that a train comes to a halt
- Hard- and software: embedded system
- key safety critical component

An Odometer

- The physics:



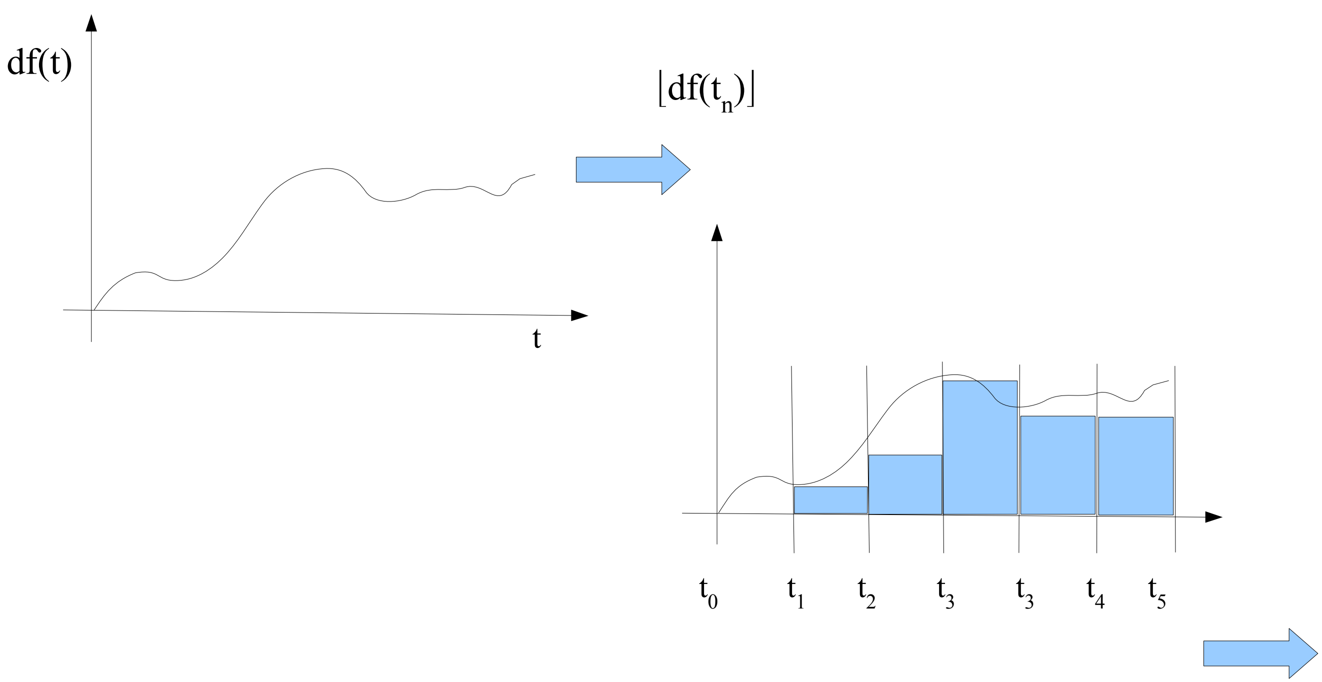
An Odometer



- Movement detection by gear wheels

An Odometer

- Movement, its detection and encoder sequences



Encoder Sequence:

101,100,010,110,110,...

Problem: Get An Odometer Formally Certified

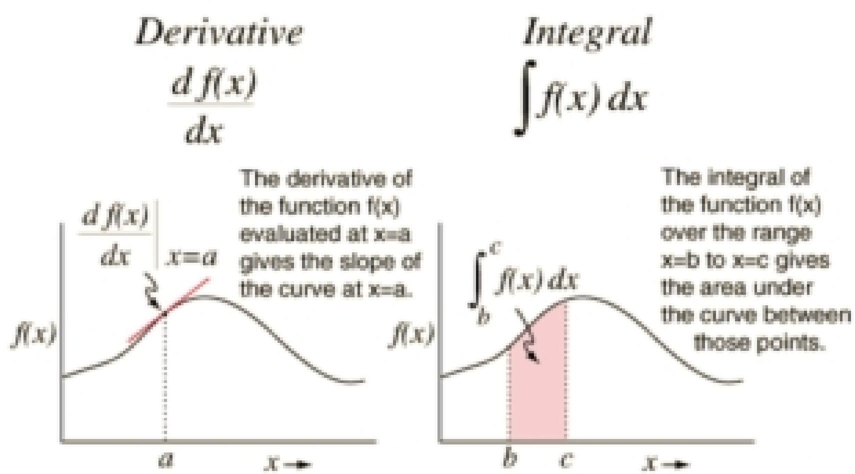
- Certification Critical Components
  - Safety in Railways: CENELEC 50126/50128
  - Safety in Avionics : DO 178 B/CSecurity:
  - COMMON CRITERIA (ISO 15408)
- Goal: Complete Traceability of Development, Hypothesis and Assumptions of Models, and Evidence
- Formal Methods recommended or mandatory

Verification Methdology

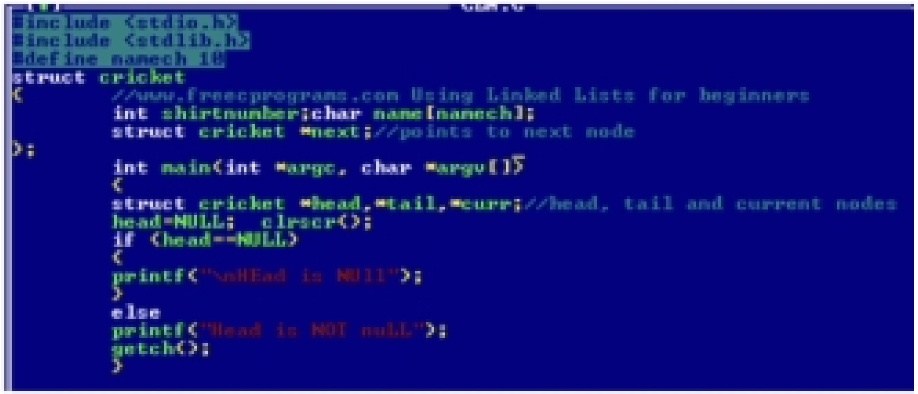
- We use Isabelle (http://isabelle.in.tum.de) for the formal development process

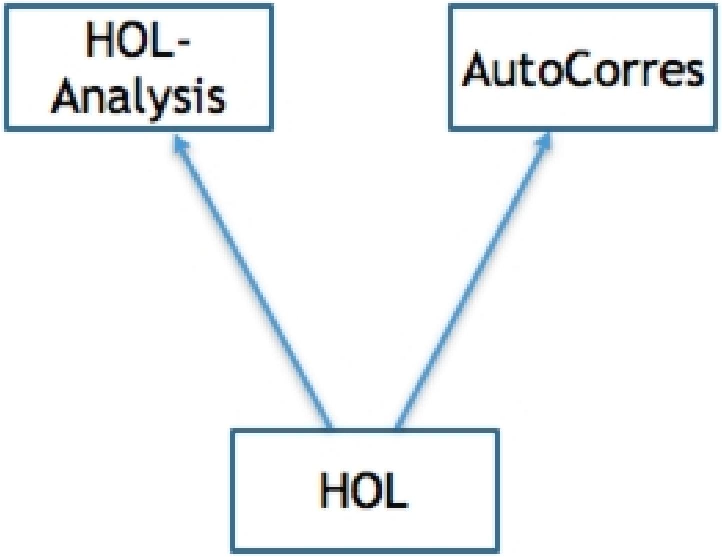
- Isabelle: The "Eclipse" of Formal Methods
  - offering plugin mechanism
  - an Prover IDE
  - code-generators
    (SML(->C), OCaml (-> FSharp, dotnet), Haskell, Scala)
  - documentation generator
  - modeling methodology for Higher-Order Logic
  - language for automated and interactive proof

Verification Methdology

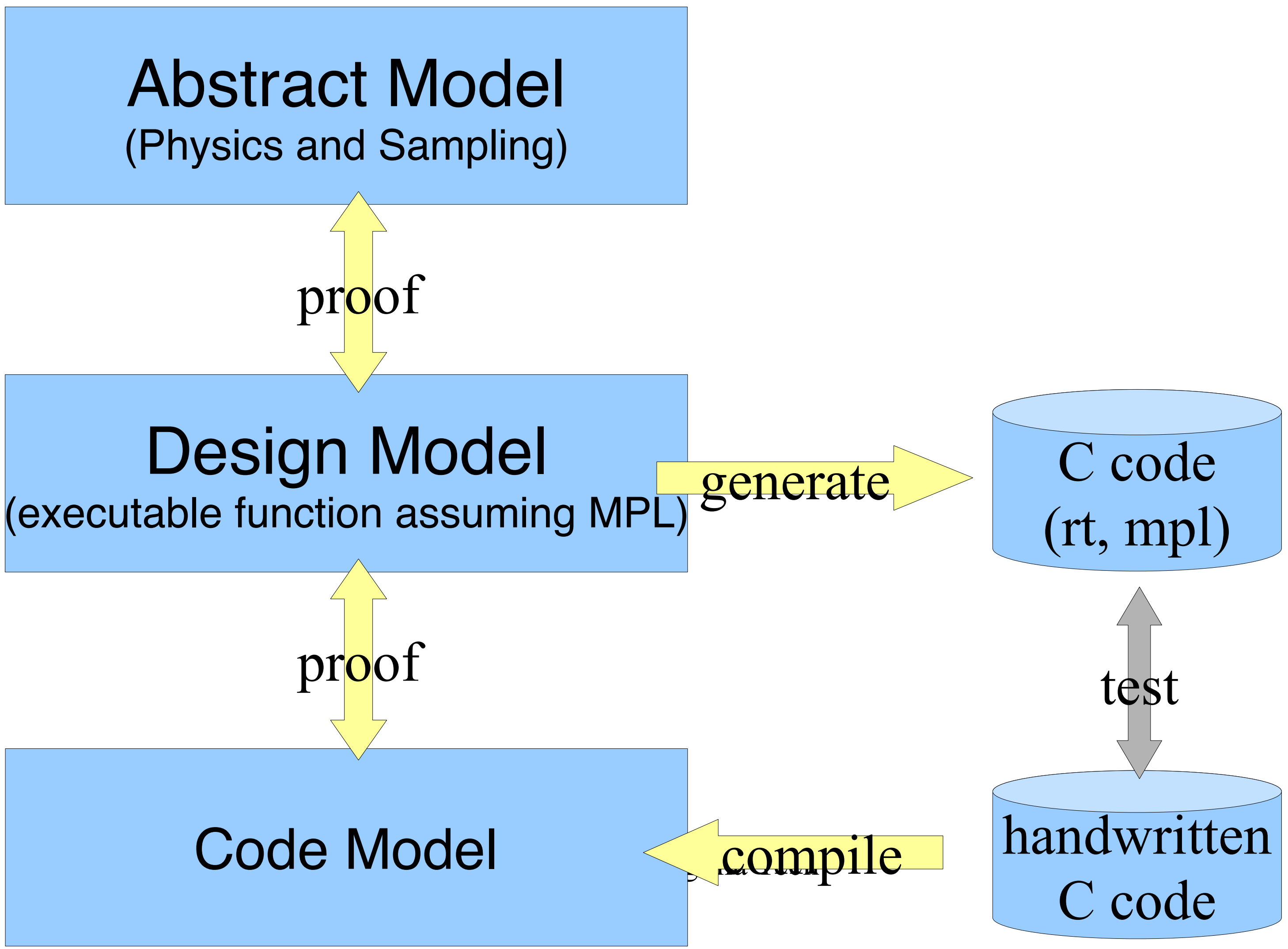


Isabelle can handle both mathematical and program verification proofs



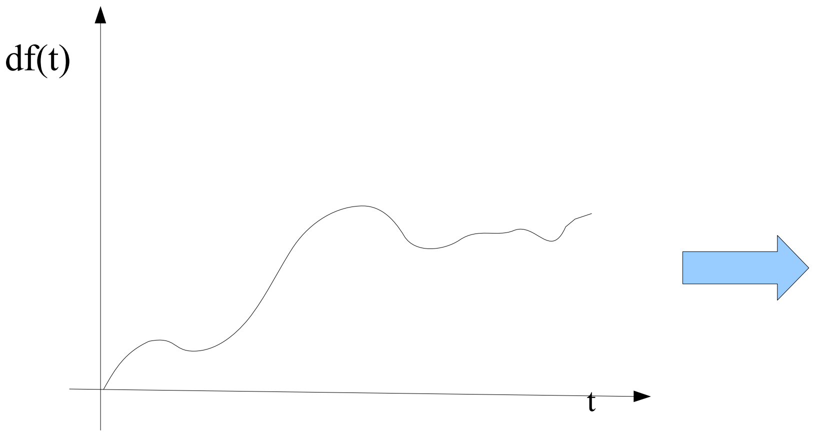


Verification Methodology



Verification Methodology

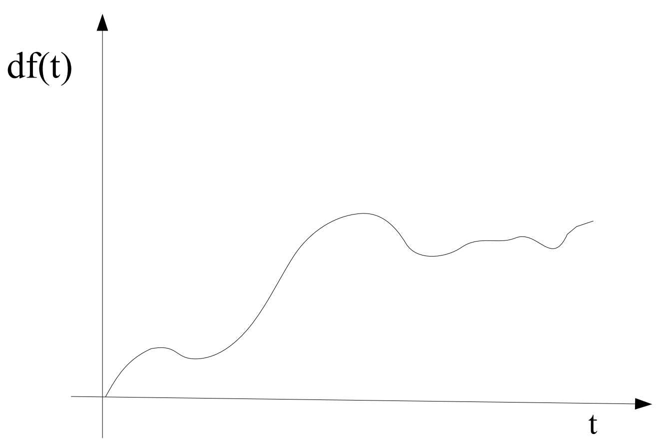
- Abstract Model:
  Requirements Definition and their Analysis
  - well-behaved distance functions:




```
type_synonym distance_function = "real ⇒ real"

definition Speed :: "distance_function ⇒ real ⇒ real"
  where   "Speed f ≡ deriv f"

definition Acceleration :: "distance_function ⇒ real ⇒ real"
  where "Acceleration f ≡ deriv (deriv f)"

definition Jerk :: "distance_function ⇒ real ⇒ real"
  where   "Jerk f ≡ deriv (deriv (deriv f))"
```

Verification Methodology

- Abstract Model:
  Requirements Definition and their Analysis
  - well-behaved distance functions:




```
definition normally_behaved_distance_function :: "(real ⇒ real) ⇒ bool"
  where "normally_behaved_distance_function df =
                    ( ∀ t. df(t) ∈ ℝ≥0 ∧
                     (∀ t ∈ ℝ≤0. df(t) = 0) ∧
                     df differentiable onℝ ∧
                     (Speed df) differentiable onℝ ∧
                     (∀ t. (Speed df) t ∈ {-SpeedMax .. SpeedMax}) ∧
                     (Acceleration df) differentiable onℝ ∧
                     (∀ t. (Acceleration df) t ∈ {-¦AccelerationMax¦ .. ¦
                    )"
```

Verification Methodology

- Abstract Model:
  Requirements Definition and their Analysis
  - shaft encodings:

Phase	C1	C2	C3
1	0	0	1
2	1	0	1
3	1	0	0
4	1	1	0
5	0	1	0
6	0	1	1
7	0	0	1
8	1	0	1
9	1	0	0
10	1	1	0
11	0	1	0
12	0	1	1

```
fun phase0 :: "nat ⇒ shaft_encoder_state"   where
       "phase0 (0) =       ⦇ C1 = False, C2 = False, C3 = True ⦈"
      |"phase0 (1) =       ⦇ C1 = True,  C2 = False, C3 = True ⦈"
      |"phase0 (2) =       ⦇ C1 = True,  C2 = False, C3 = False⦈"
      |"phase0 (3) =       ⦇ C1 = True,  C2 = True,  C3 = False⦈"
      |"phase0 (4) =       ⦇ C1 = False, C2 = True,  C3 = False⦈"
      |"phase0 (5) =       ⦇ C1 = False, C2 = True,  C3 = True ⦈"
      |"phase0 x   =       phase0(x - 6)"

definition Phase :: "nat ⇒ shaft_encoder_state"
  where    "Phase (x) =  phase0 (x-1) "
```

Verification Methodology

- Abstract Model:
  Requirements Definition and their Analysis
  - some simple proofs on safety:

```
lemma Encoder_Property_1: "(C1(Phase x) ∧ C2(Phase x) ∧ C3(Phase x)) = False"
  proof (cases x)
    case 0 then show ?thesis  by (simp add: Phase_def)
  next
    case (Suc n) then show ?thesis
      by(simp add: Phase_def,rule_tac n = n in cycle_case_split,simp_all)
  qed

lemma cycle_mod : " phase₀ x = phase₀(x mod 6)"
  apply(subst mod_div_mult_eq[symmetric, of _ 6])
  using phase₀_is_cycle by blast

lemma phase₀_inj_on_6: "∀x<6. ∀y<6. phase₀ x = phase₀ y ⟶ x = y"
```

Verification Methodology

- Abstract Model:
  Requirements Definition and their Analysis

  - definition of sampling of a distance function:

```
definition encoding :: "distance_function ⇒ nat ⇒ real ⇒ shaft_encoder_state"
  where   "encoding df init_enc_pos  == λx. Phase(nat⌊df(x) / δs_res⌋ + init_enc_pos)"
```

Verification Methodology

- Abstract Model:
  Requirements Definition and their Analysis
  - theorem: sampling is accurate for well-behaved distance functions:

```
theorem no_loss_by_sampling :
  assumes * : "normally_behaved_distance_function df"
    and  ** : "δt_odo * Speed_Max < δs_res"
             (* This establishes a constraint between  w_circ,
                tpw, Speed_Max and sample_frequency *)
  shows      "∀ δt≤δt_odo. 0<δt ⟶
                   (∃f::nat⇒nat.
                         retracting f ∧
                         sampling df init_enc_pos δt = (sampling df init_enc_pos  δt_odo) o f)"
```

Verification Methodology

- Abstract Model: Requirements Definition and their Analysis
  - theorem: sampling is accurate for well-behaved distance functions:

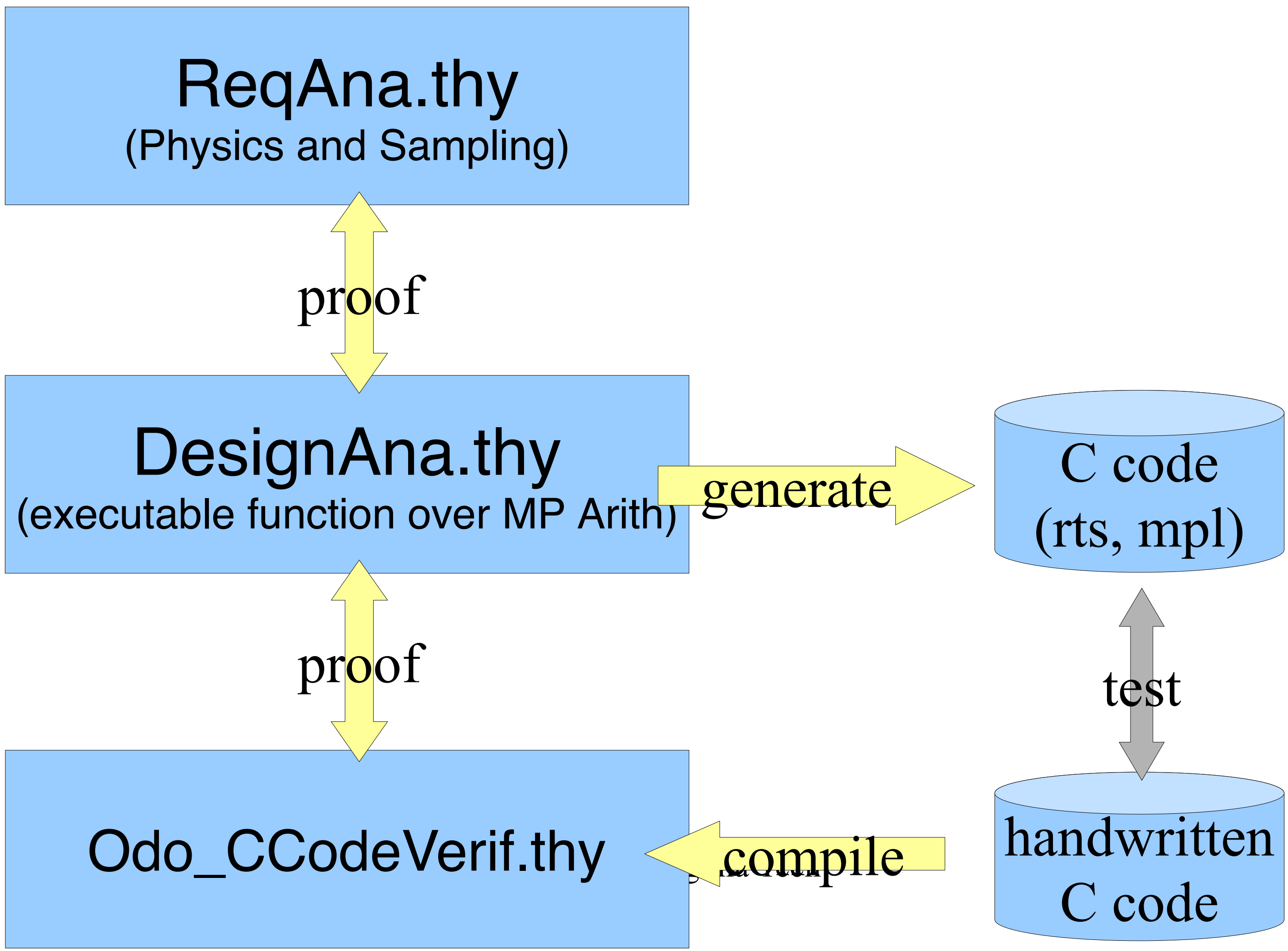
PROOF : Nothing for the faint-hearted ...

Verification Methodology

- Abstract Model:
  Requirements Definition :
  - input and output of the module:

```
record "output" =
          Odometer_Status                  :: boolean
          Odometric_Position_Valid         :: boolean
          Odometric_Position_Count         :: unsigned_int_32_bit
          Odometric_Position_TimeStamp     :: unsigned_int_32_bit
          Last_Marker_Position             :: unsigned_int_32_bit
          Last_Marker_TimeStamp            :: unsigned_int_32_bit
          Relative_Position                :: unsigned_int_32_bit
          Speed₀                           :: signed_int_32_bit
          Acceleration₀                    :: signed_int_32_bit
          Jerk₀                            :: signed_int_32_bit
          Cinematics_TimeStamp             :: unsigned_int_32_bit
```

Verification Methodology



Verification Methodology

- Requirement Analysis:
  Results:
  - Establishment of the dictionary of the physical system,
  - the principles of sampling into encoder sequences,
  - and the interface of the module.

  - main theorem establishes conditions under which the sampling can be valid in principle. („no jumps in sequence“)

Verification Methodology

- Design Analysis:
  Results:
  - Computable definitions for odo_{step} which is the heart of the odometric calculations.
  - The main theorem establishes that odo_{step} indeed approximates distance, speed and acceleration assuming a rational arithmetic with unlimited precision.
  - odo_{step} is converted into executable code as a reference for precision tests.

Verification Methodology

- C-Code Verification:
  Results:
  - We provide a handwritten C function and verify it via the C-to-HOL compiler against the design odo_{step}
  - The main theorem establishes that the C-level calculations done on bounded machine arithmetics indeed approximate the calculations of odo_{step} under certain conditions.
  - This proof work just started.

Certification Methodology

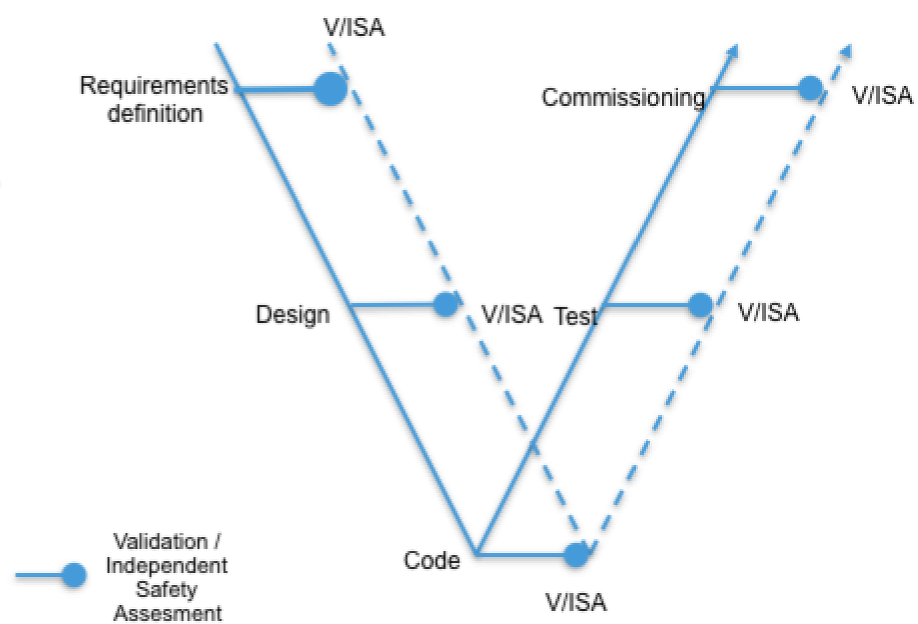
- Observation: Formal Models are not Enough for Formal Certification

Certification Methodology

- Observation: Formal Models are not Enough for Formal Certification

Software certification

- The railway industry requires certification processes to be applied to ensure the safety of transportation systems
  - CENELEC
- Software certification as a W process can be slippy, long and expensive

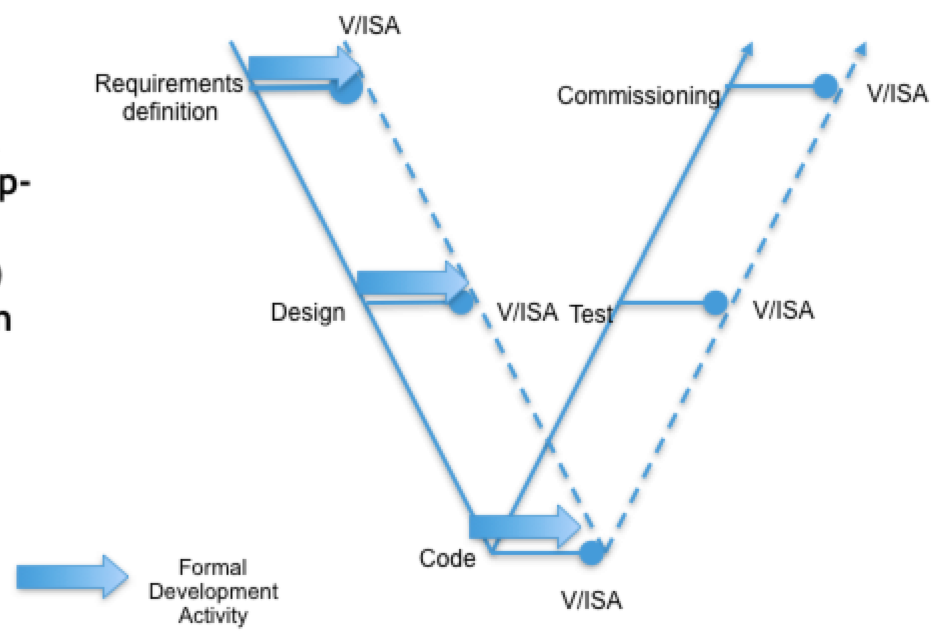


Certification Methodology

- Observation: Formal Models are not Enough for Formal Certification

- The railway industry requires certification processes to be applied to ensure the safety of transportation systems
  - CENELEC

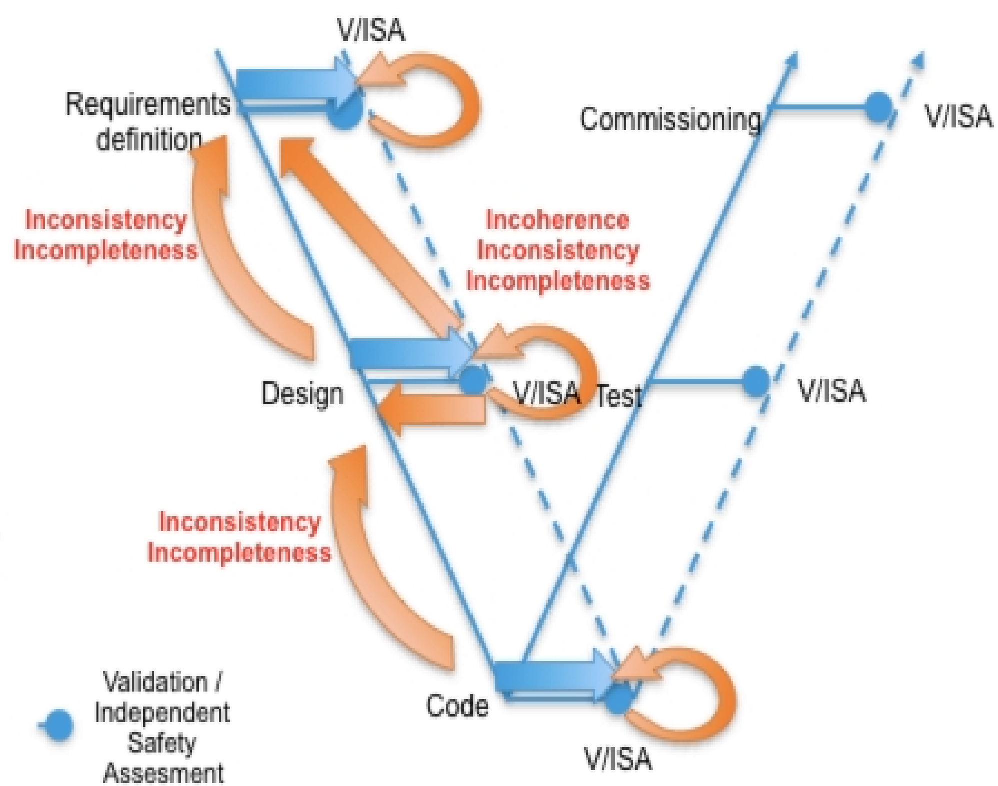
- Software certification is a "deeper" develop-ment process
- Document 10-40 times larger than the primary artefact
- Document Development must be engineered



Certification Methodology

- Observation: Formal Models are not Enough for Formal Certification

- Software certification as a W process can be slippy, long and expensive
- Some Form of "agility" is needed!



Certification Methodology

- CVCE Methodology
  - Logical Consistency
  - ... and Coherence between
    semi-formal and formal evidence (tests, proofs)
  - ... our experience shows, that
    document coherence and traceability
    is a major cost problem in certifications

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - Start informal requirements capture within Isabelle

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - Start informal requirements capture within Isabelle

```
text{*
Accurate information of train's location along a track is crucial to safe railway operation.
Position measurement along a track infrastructure usually lays on a set of independent measurements
based on different physical principles - as a way to enhance precision and availability.  As a rule,
the train gets absolute position coordinates by running over stationary markers in the track, while
an odometer allows estimating a relative location while the train runs between successive markers.
The proposed use case comprises two services:
▪ Odometrics module, which processes the signals issued by an incremental
  shaft encoder attached to a bogies axle, producing a real-time estimation of the trains progress.
▪ Kinematics module, which calculates:
  ▸ the trains relative position
  ▸ the trains absolute speed, acceleration and jerk.
*}
```

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - Start informal requirements capture within Isabelle
    - ... add formalizations of key concepts early ("literate programming style")

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - Start informal requirements capture within Isabelle
    - ... add formalizations of key concepts early

```
fun phase0 :: "nat ⇒ shaft_encoder_state"   where
       "phase0 (0) =       ⦇ C1 = False, C2 = False, C3 = True ⦈"
      |"phase0 (1) =       ⦇ C1 = True,  C2 = False, C3 = True ⦈"
      |"phase0 (2) =       ⦇ C1 = True,  C2 = False, C3 = False⦈"
      |"phase0 (3) =       ⦇ C1 = True,  C2 = True,  C3 = False⦈"
      |"phase0 (4) =       ⦇ C1 = False, C2 = True,  C3 = False⦈"
      |"phase0 (5) =       ⦇ C1 = False, C2 = True,  C3 = True ⦈"
      |"phase0 x   =       phase0(x - 6)"

definition Phase :: "nat ⇒ shaft_encoder_state"
  where   "Phase (x) =  phase0 (x-1) "
```

Certification Methodology

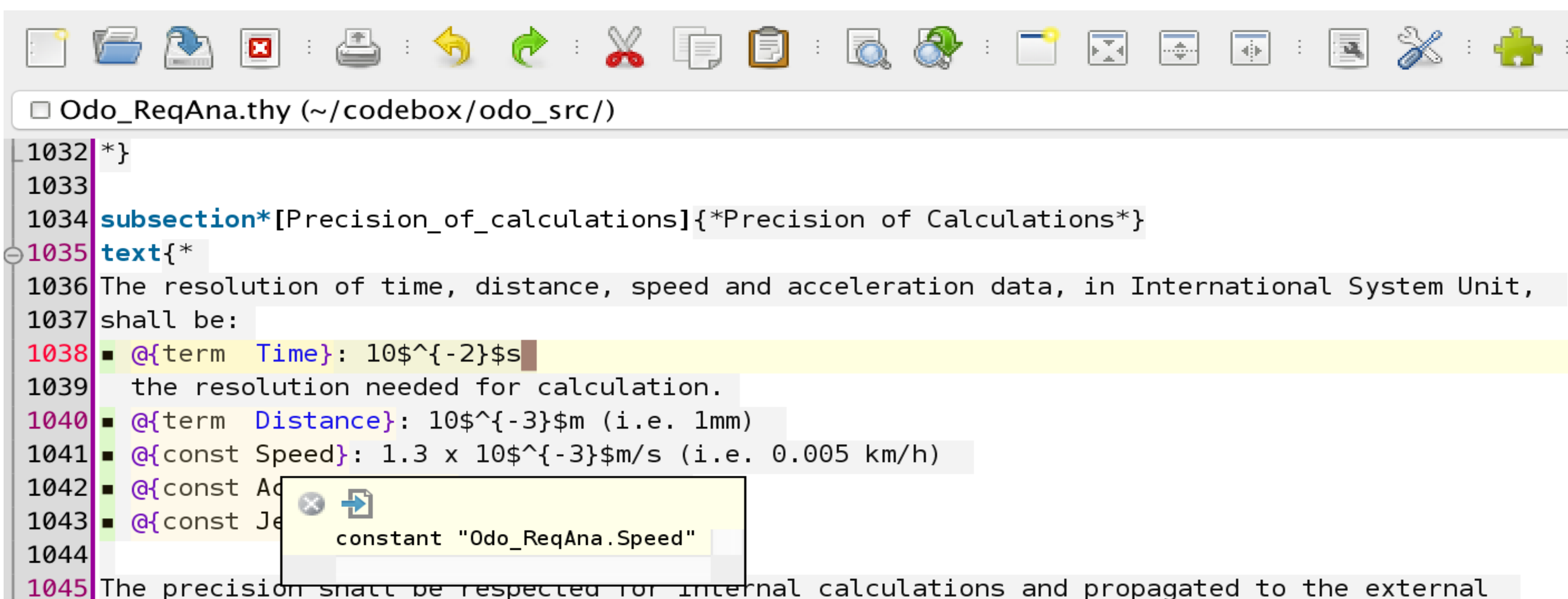
- Development Method
  - Versioning of all artefacts, integrate into global document

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - Start informal requirements capture within Isabelle
    - ... add formalizations of key concepts early
    - make the informal semi-formal: highlight antiquotations

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - Start informal requirements capture within Isabelle
    - ... add formalizations of key concepts early
    - make the informal semi-formal: highlight antiquotations



Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc's inside Isabelle
    - use a certification specific ontology to enforce links as antiquotations.
    - ... turn links into antiquotations

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc’s inside Isabelle
    - use a certification specific ontology to enforce links as antiquotations.
    - ... turn links into antiquotations

definition of a „software related application constraint“

```
text*[enough_samples::srac]{* Note that the theorem above establishes a constraint between
@{consts W_circ}, @{consts tpw} , @{consts Speed_Max} and  sample_frequency; since this
exported constraint is fundamental for the safe functioning of  odometer and therefore
a safety-related exported application constraint. It is formally expressed as follows:
*}
```

Certification Methodology

- Development Method
  - Versioning of all artefacts, integrate into global document
  - Make doc’s inside Isabelle
    - use a certification specific ontology to enforce links as antiquotations.
    - ... turn links into antiquotations

applications of a „srac“ ref as an exported „exported constraint“. Compatibility via „is_A“ relation in the CENELEC Ontology.

```
text{* Summing up, the property that the odometer provides sufficient sampling
precision --- meaning no wheel encodings were ``lost'' compared to any sampling done with
a higher sampling rate --- can be established under the set of general hypothesis captured
in @{docref ‹general_hyps›} (formally expressed in @{thm normally_behaved_distance_function_def})
and the SRAC @{ec ‹enough_samples›} formally expressed by @{thm srac1_def}. *}
```

Certification Methodology

- Development Method

  - Semi-formal Requirements capture the ontology framework enforces for CENELEC

    - tracking of assumptions, hypothesis, constraints
    - definitions, theorems
    - code
    - tests
    - the structure and usage of links.

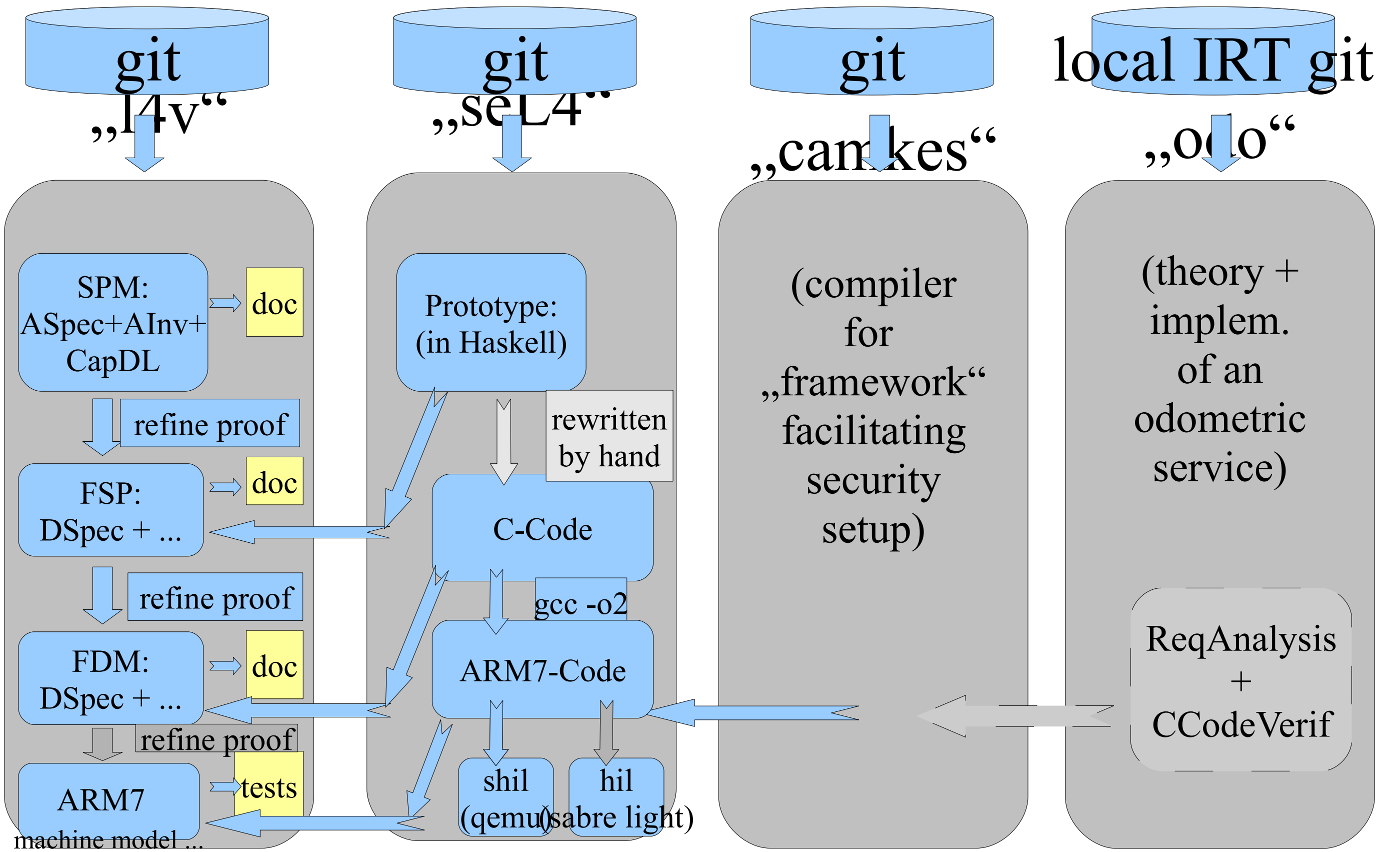
Integrating into seL4-OS

seL4: secured L4 (Klein & Heiser SOSP'09)

- OS Kernel in the L4 tradition
- advanced Security (Access-Control) Model
  "Take-Grant Capabilities"
- virtual memory, dyn. thread creation,
  IPC, Fast-Track-IPC, support of AnoCom.
- designed to be formally verifiable (in Isabelle/HOL)
- designed to be performant

Integrating into seL4-OS

- Scaling up: Integrating Odo into seL4



CVCE : An Environment for Formal “Agile Development“

- CVCE : Continuous Verification and Certification Environment
  - Isabelle/HOL: core for consistency
  - Global Version Management
  - Global Config Management (docker)
  - jenkins

- CVCE - jenkins view (I):

Conclusion

- Formal Development based on ITP technology is at the brink to leverage formally verified embedded subsystems
- Embracing formality can increase the agility of the development („embrace change“)
- Linking the Formal and Semi-Formal is Key to lower the costs of Formal Certifications
- SE Infrastructure (like CVCE) is Key to scale up.

Thank you.

Formal “Agile Development“

+ adaptive planning,
+ evolutionary, distributed development,
+ early delivery,
+ continuous improvement, continuous build, and
+ rapid and flexible response to change

Techniques / Methods:
- social engineering, stand-ups, pairprogramming,
- scrum sprints etc ...
- animosity of documentation, over-emphasis of tests
- see B. Meyer‘s book critical resumee (Agile! The Good, the Hype and the Ugly ...

Experimental Evaluation

- in more detail:

Bugs found

during testing: 16

during verification:

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

Experimental Evaluation

- implem errors covered in more detail:

Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion

Evaluation

- cost analysis
  - overall : 25 py investment, mostly for the refinement proof
  - about 10 py infrastructure (reusable?)
  - arguably cost effective:

Effort

Haskell design	2	py
First C impl.	2	weeks
Debugging/Testing	2	months
Kernel verification	12	py
Formal frameworks	10	py
Total	25	py

Cost

Common Criteria EAL6:	**$87M**
L4.verified:	**$6M**

seL4 is free - what does this mean to you ?

- seL4 became an open source project
  (see video https://www.youtube.com/watch?v=lRndE7rSXiI)

The seL4 Microkernel

Security is no excuse for poor performance!

seL4 Security. Performance. Proof.

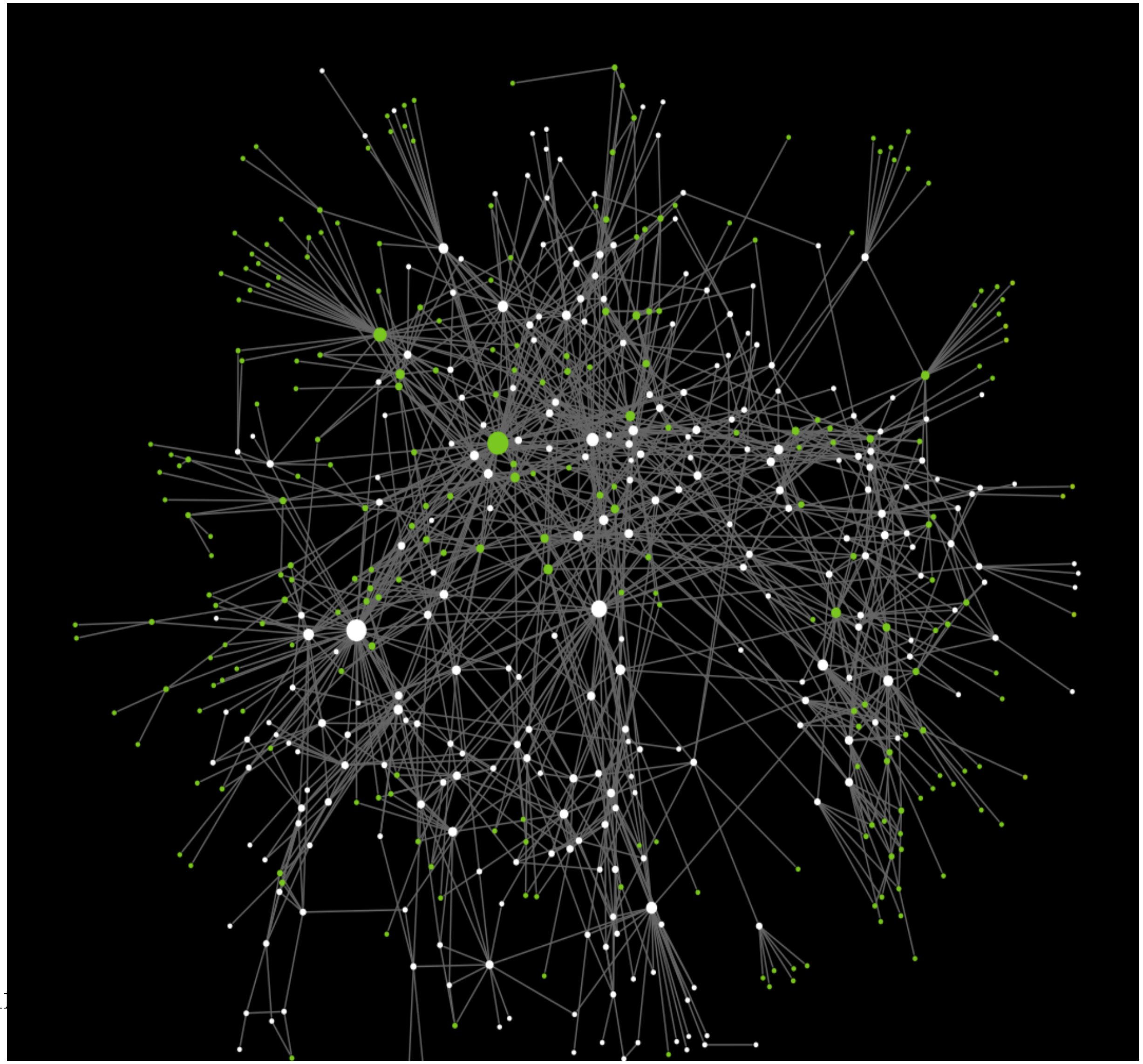
The world's first operating-system kernel with an end-to-end proof of implementation correctness and security enforcement is available as open source.

Sign up to sel4-announce | Sign up to sel4-devel

How to get it | on GitHub | FAQ

seL4 is free - what does this mean to you ?

- anybody can contribute
  (and chances of acceptance are high if proof provided)
- consistency can be maintained even in distributed collaboration
  (easy impact analysis in Isabelle)



Verification Methodology

- Supported C this way:

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
    case (word_t)SchedulerAction_ResumeCurrentThread:
        break;

    case (word_t)SchedulerAction_ChooseNewThread:
        chooseThread();
        ksSchedulerAction = SchedulerAction_ResumeCurrentThread;
```

Everything from C standard

- **including**:
  - pointers, casts, pointer arithmetic
  - data types
  - structs, padding
  - pointers into structs
  - precise finite integer arithmetic

- **minus:**
  - goto, switch fall-through
  - reference to local variable
  - side-effects in expressions
  - function pointers (restricted)
  - unions

- **plus** compiler assumptions on:
  - data layout, encoding, endianess

```
if(!isRunnable(thread)) {
    next = thread->tcbSchedNext;
    tcbSchedDequeue(thread);

else {
    switchToThread(thread);
    return;
```

Verification Methodology

- Final step :
  Eliminate C - 2 - Isabelle/HOL/Simpl

  - generated optimized ARM assembly (conventionally via gcc -o4 ...)
  - re-use an ARM operational semantics model(going back to A. Fox)
  - use smt technology to verify that
    action contracts are still valid on machine level ...