Against (Formal) Method?

An experience report on Formal Methods from a developer point-of-view

Arnaud Bailly - @abailly.bsky.social

Pankzsoft

2025-03-14

Introduction

Agenda

- Introduction
- Context & Experiments
- Findings & Analysis
- Takeaways & Conclusion

Where do I speak from?

- Dev/Tech Lead/Architect/Consultant for 30+ years
- PhD in computer science (20 years ago)
- Dedicated *eXtreme Programming* Practitioner
- *Cautious believer* in the benefits of formal methods
- Experience limited to *specific* types of software

Too Long; Didn't Stay

Formal Methods (FM) are not a *Silver Bullet* but a useful tool that can bring value to most software development efforts

- *Proving software correctness* is still out of reach for most teams and systems
- FMs can be introduced incrementally in the *Software Development Lifecycle* (SDLC)
- FM can help grow and maintaing a powerful *Ubiquitous Language*

Context & Experience

Why use Formal Methods?

- Fun: Because it's so cool...
- Computer science: Study type systems, mathematics, programming languages, etc.
- Applied science: Back research with machine-checkable proofs of stated properties
- Software quality: Provide strong safety guarantees *make the software right*
- Software design: **Improve design with better models** *make the right software*

Cardano

Key features:

- Globally distributed and fully decentralized *open* system w/ 3000+ block producing nodes and 100s of *developers*
- *Security & safety are critically important*
- Established tradition of working with Formal Methods
- Research plays a key role in the system's development

R&D Projects

Projects I worked on had a common theme:

- More or less (more) complex algorithms and protocols w/ proven properties
- Written by cryptographic & security researchers, aka. *mathematicians*, with heavy proof apparatus
- Require collaboration of people with diverse background and skills
- Strong safety and/or liveness requirements

How do we turn research papers into reliable working software?

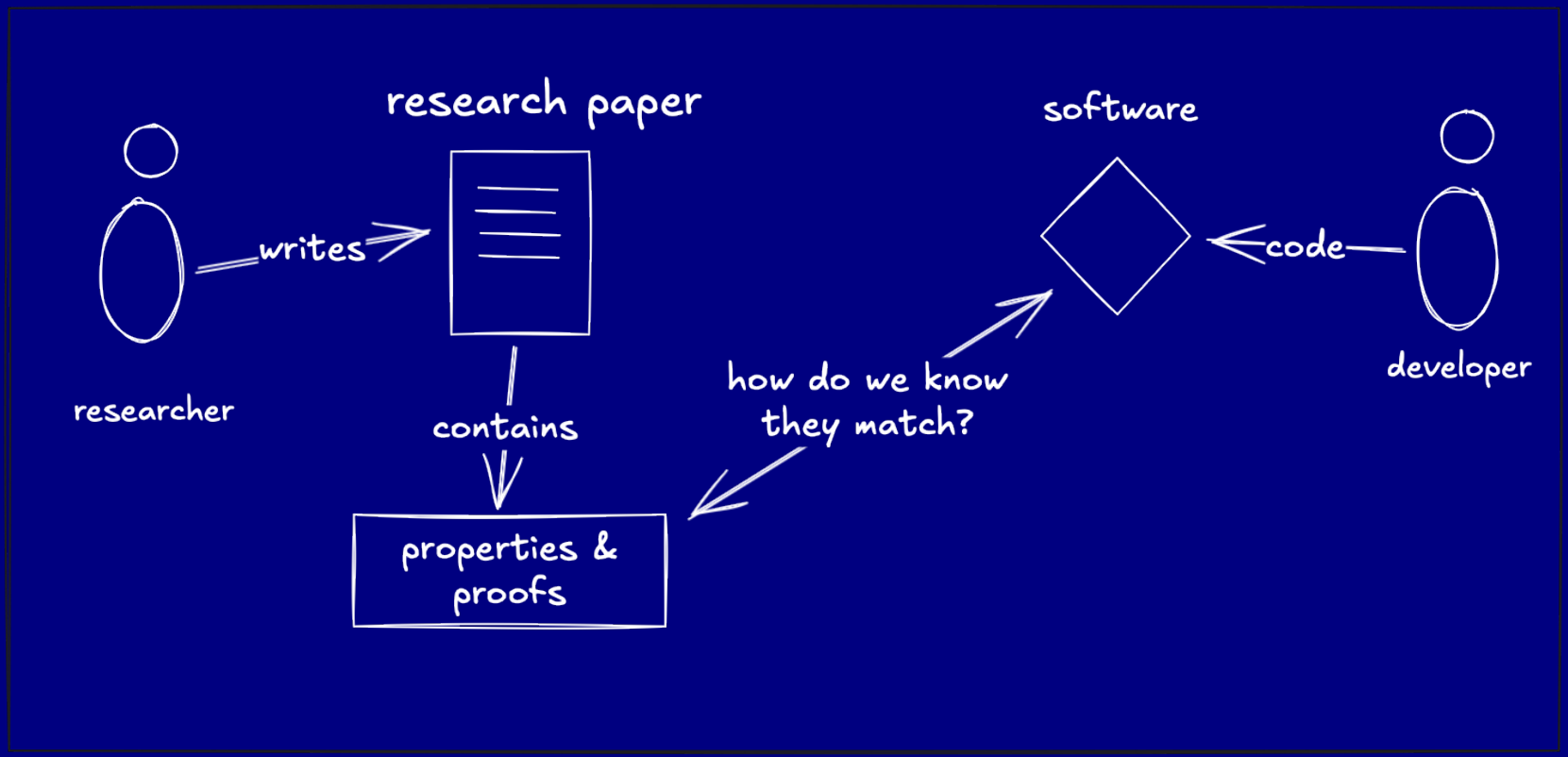


Figure 1: Relating Proofs & Programs

Researchers are Domain experts

Peras

- One project within *Innovation streams*
- Experiment and refine structured *method* to go from research ideas to products
- Small (3.5 people) team: Researcher, FM engineer, 2 x Architects/Developers

Process & Tools

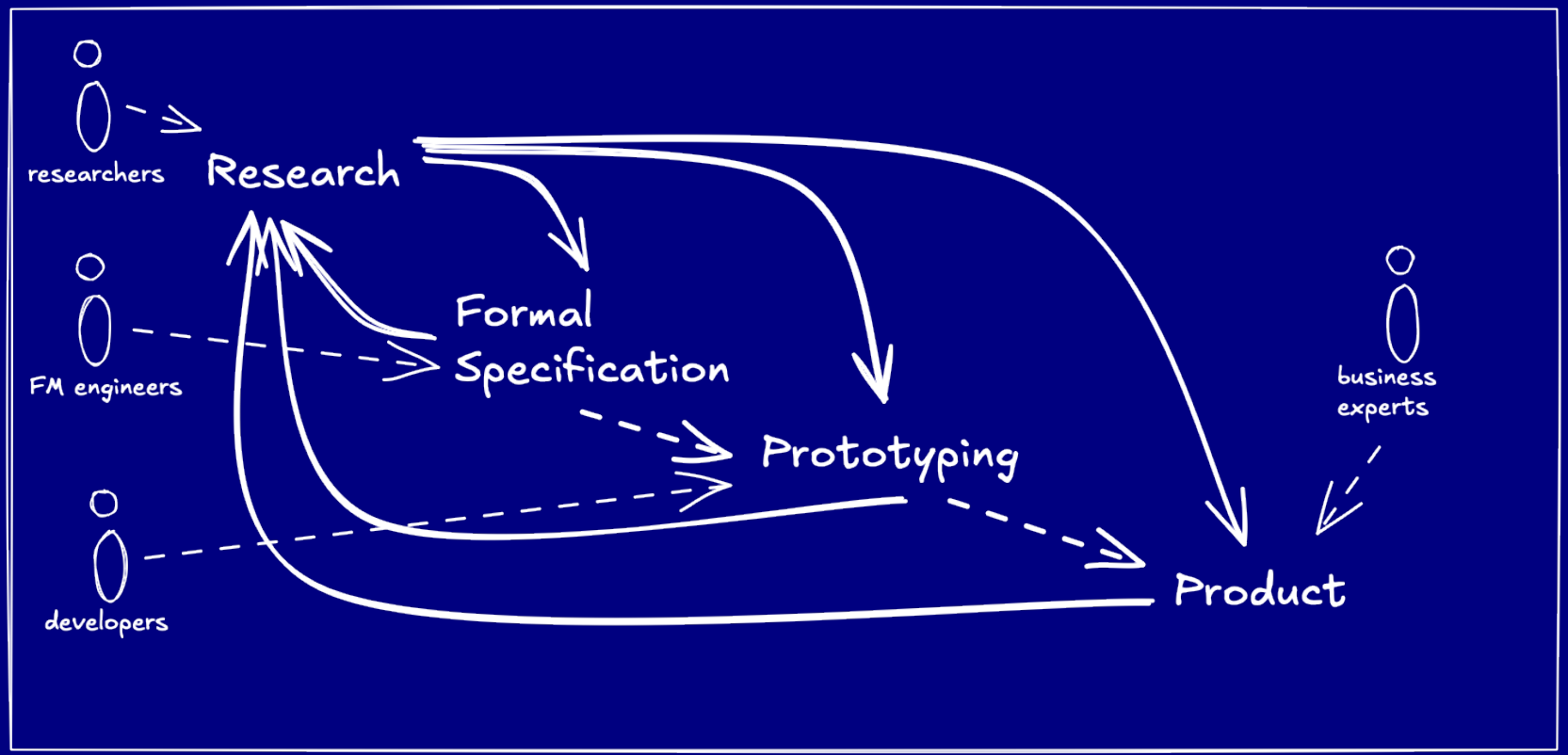


Figure 2: Peras workflow

- ΔQ: Network performance formalism
- *Agda*: Formal specification language
- *Agda2HS*: Generate Haskell code from Agda specification
- *quickcheck-dynamic*: Haskell code to generate conformance tests
- *Haskell* and *Rust*: Target languages for prototypes

Agda as specification language

- Protocol modelled in Agda using Small-steps semantics specifying the impact of each node "actions" on global state
- Took inspiration from previous work on Formalizing Nakamoto-Style Proof of Stake in Coq
- Heavy emphasis on producing a *readable specification*

```
module Peras.SmallStep where
```

Small-step semantics

The small-step semantics of the **Ouroboros Peras** protocol define the evolution of the global state of the system modelling *honest* and *adversarial* parties. The number of parties is fixed during the execution of the protocol and the list of parties has to be provided as a module parameter. In addition the model is parameterized by the lotteries (for slot leadership and voting committee membership) as well as the type of the block tree. Furthermore adversarial parties share generic, adversarial state.

References:

- Formalizing Nakamoto-Style Proof of Stake, Søren Eller Thomsen and Bas Spitters

Parameters

The parameters for the *Peras* protocol and hash functions are defined as instance arguments of the module.

```
module _ ⦃ _ : Hashable Block ⦄
         ⦃ _ : Hashable (List Tx) ⦄
         ⦃ _ : Params ⦄
         ⦃ _ : Network ⦄
         ⦃ _ : Postulates ⦄

         where
```

Figure 3: Agda Specification

Agda driving conformance tests

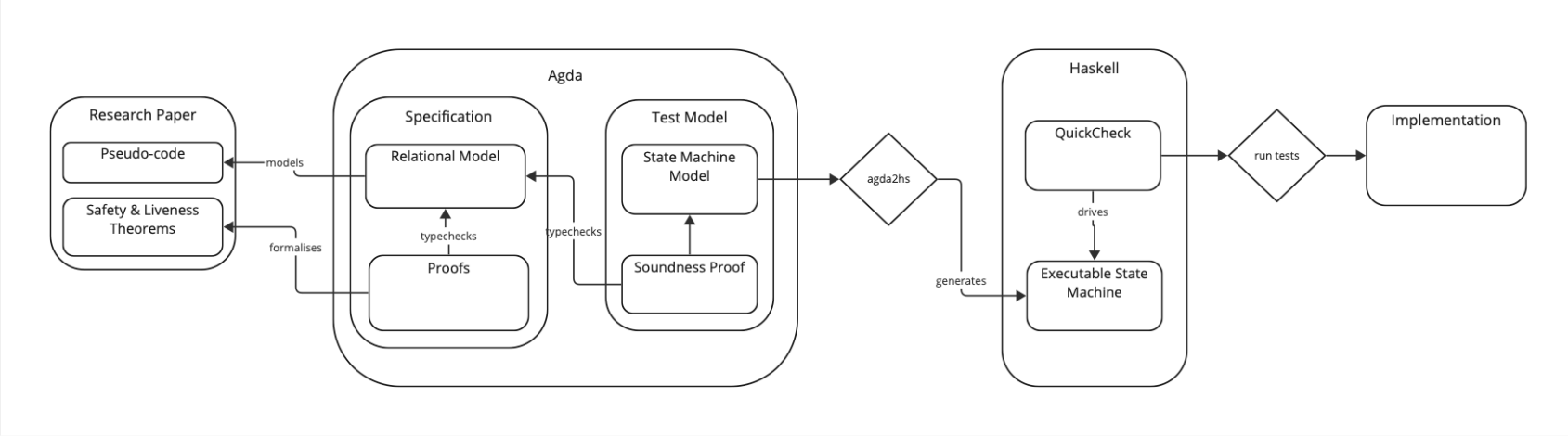


Figure 4: Peras testing

Outcomes

What went well

A Better Standard

Literate Agda formed the backbone of a Cardano Improvement Proposal standard specification.

Figure 5: CIP-140

Improved Feedback loop

- Formalisation (and prototyping) uncovered shortcomings in the protocol that lead to improvements
- Interaction of formal modeling and prototyping uncovered a few bugs in *both*
- Having a “small” formal model helped bootstrap development beyond prototyping

Towards a "Security Research" DSL

Voting and Block Creation

Parties P vote and create blocks as follows:

```
upon entering new slot s
        if P is leader in slot s
                B := new block extending C_pref
                if Certs[r_current-2] = null
                   and r_current - round(cert_seen) <= A
                   and round(cert_seen) > round(cert_chain)
                       B := (B, cert_seen)
                C_pref := C_pref || B
                output (chain, C_pref<-W>) to Z
```

Figure 6: "Informal" pseudocode

Variables

```
Cpref : Chain                          -- Preferred Peras chain
certseen : Certificate            -- Latest certificate in Certs
certchain : Certificate          --  Latest certificate on Cpref
Vpref : List (set Vote)      -- Pref. vote sets (idx'd by rd no)
Certs : List Certificate -- Pref. certificates (idx'd by rd no)
```

Below, it is assumed that `certseen` and `certchain` are updated automatically when `Certs` or `Cpref` change.

Voting and Block Creation

Parties P vote and create blocks as follows:

```
onNewSlot : Party -> Slot -> Node ⊤
onNewSlot p s =
 when (p isLeaderInSlot s) (do
     b ← forgeBlock Cpref
     let rcurrent = s / U
     when
       ( (Certs [ rcurrent − 2 ]) == null
           ∧ ((rcurrent − round certseen) ≤ A)
           ∧ (round certseen > round certchain)
       ) do
         let b' = Certify b certseen
         Cpref ← Cpref extendWith b'
         output (Cpref trimmedBy W))
```

Figure 7: “Formal” pseudocode

What could be improved

Silos

- Integrating FM engineering in the day-to-day activity of the team is not straightforward
- FM engineering is a specialty that's not (yet) widespread
- Creating silos is a slippery slope that leads to *DBA* or *Ivory Tower architects* situations

Coping with change

CP.v

```
2145 { rewrite (@super_slots_blocks _ _ N) //; [| by rewrite addn1| by rewrite subn1 leq_pred].
2146   rewrite -2!size_cat -2!(size_map (pos^~ N)).
2147   apply/uniq_leq_size.
2148   (** Uniqueness of SBs in world-tree *)
2149   { rewrite /=.
2150     apply/subseq_uniq; [|apply/(unique_sb_pos N0N) => //].
2151     by apply/map_subseq/filter_subseq. }
2152   (** Subset of positions. *)
2153   move=> sbpo/mapP [] sb + -> {sbpo}.
2154   rewrite /super_blocks_world_range mem_filter.
2155   move/andP=> [] time=> sbin /=.
2156   rewrite 2!map_cat.
2157   set bh := block_history _.
2158   (* Establishing positions in cs1 and c1 *)
2159   move/(@cfb_map_iota _ bh): (best_chain_valid (tree l1) (t_now N -1)).
2160   do 2!apply: instant1=> //.
2161   { apply/subset_trans; first by apply/best_chain_in_all.
2162     apply/subset_trans; first by apply/filter_subset.
2163     apply/subset_trans; first by apply/(subset_honest_tree N0N honest_p1).
2164     by apply/honest_tree_gb_history_subset. }
2165   rewrite -/bc1 -bc1_layout -/cs -cs_layout cat1s.
2166   have ->: (| c1 ++ cs1 ++ b' :: cs2 |) = (| (b' :: cs2) ++ cs1 ++ c1  |).
2167   { rewrite !size_cat addnA addnC.
2168     by apply/f_equal; rewrite addnC. }
2169   rewrite !map_cat !size_cat !iota_add !rev_cat -catA -catA=> /eqP.
2170   rewrite eqseq_cat=> [/andP []/eqP c1_io |]; last by rewrite size_map size_rev size_iota.
2171   rewrite eqseq_cat=> [/andP []/eqP cs1_io /eqP cs2_io|]; last by rewrite size_map size_rev size_iota.
2172   (* Establishing positions in c2 *)
2173   move/(@cfb_map_iota _ bh): (vc).
2174   do 2!apply: instant1=> //.
2175   { by apply/(subset_trans sub)/filter_subset. }
2176   rewrite -c_layout -/cs -cs_layout cat1s.
2177   have ->: (| c2 ++ cs1 ++ b' :: cs2 |) = (| (b' :: cs2) ++ cs1 ++ c2  |).
2178   { rewrite !size_cat addnA addnC.
2179     by apply/f_equal; rewrite addnC. }
2180   rewrite !map_cat !size_cat !iota_add !rev_cat -catA => /eqP.
2181   rewrite eqseq_cat=> [/andP []/eqP c2_io |]; last by rewrite size_map size_rev size_iota.
2182   (* Moving forward *)
2183   rewrite mem_cat cs1_io mem_rev mem_iota add1n.
2184   (* The super block was aware of b' when mined. *)
2185   have ->: |b' :: cs2 | < |cfb sb bh|.
2186   { have ->: b' :: cs2 = cfb b' bh.
2187     { apply/esym/cfb_valid_chain'=> //.
2188       - apply/(@valid_chain_short_l (c1 ++ cs1)).
2189         rewrite -catA. rewrite cs_layout bc1_layout.
2190           by apply/best_chain_valid.
2191       - apply/(subset_trans _ (honest_tree_gb_history_subset N0N)).
2192         apply/(subset_trans _ (subset_honest_tree N0N honest_p1 state_p1))=> //.
2193         apply/(@subset_trans _ _ bc1).
2194         + rewrite -bc1_layout -/cs -cs_layout 2!catA.
2195             by apply/subset_catl.
2196         + by apply/(subset_trans (@best_chain_in_all _ ((t_now N) -1) (tree l1)))/filter_subset. }
```

Figure 9: Coq Proof fragment

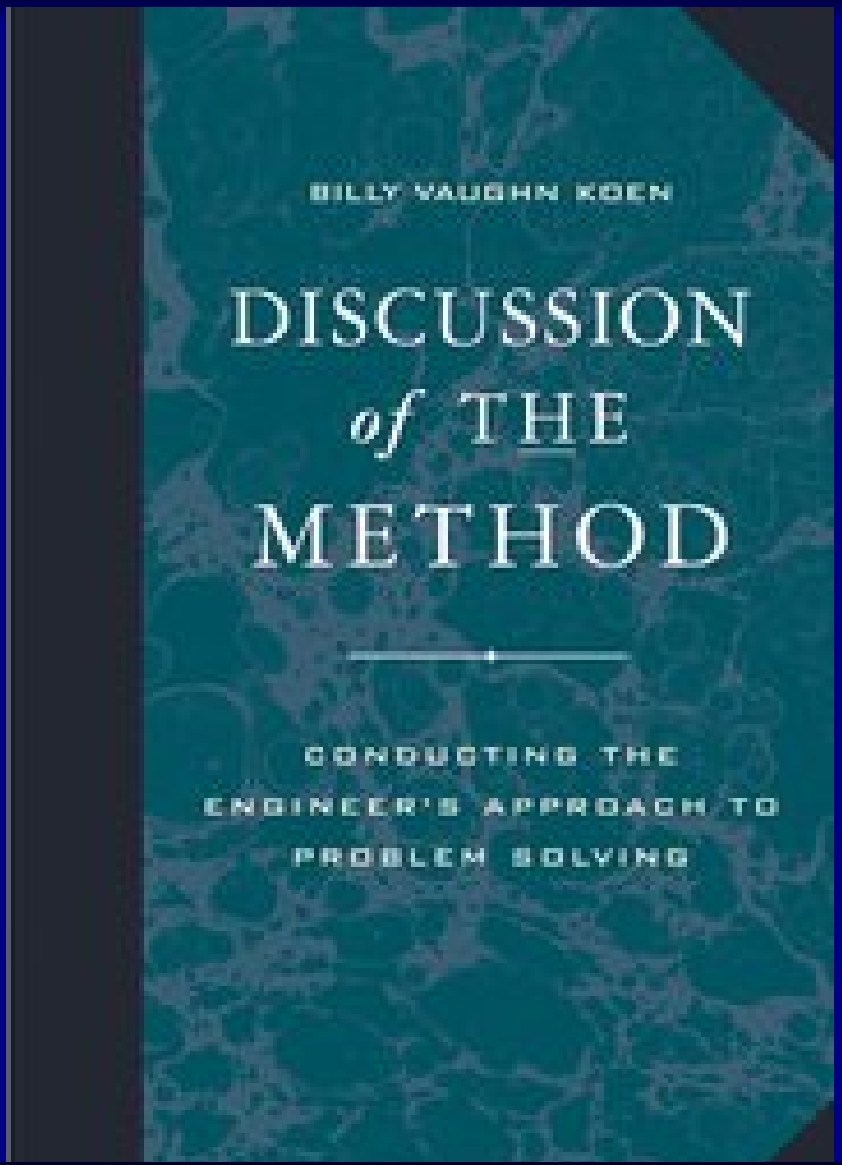
How do we keep formal specifications and FM artefacts maintenable over time?

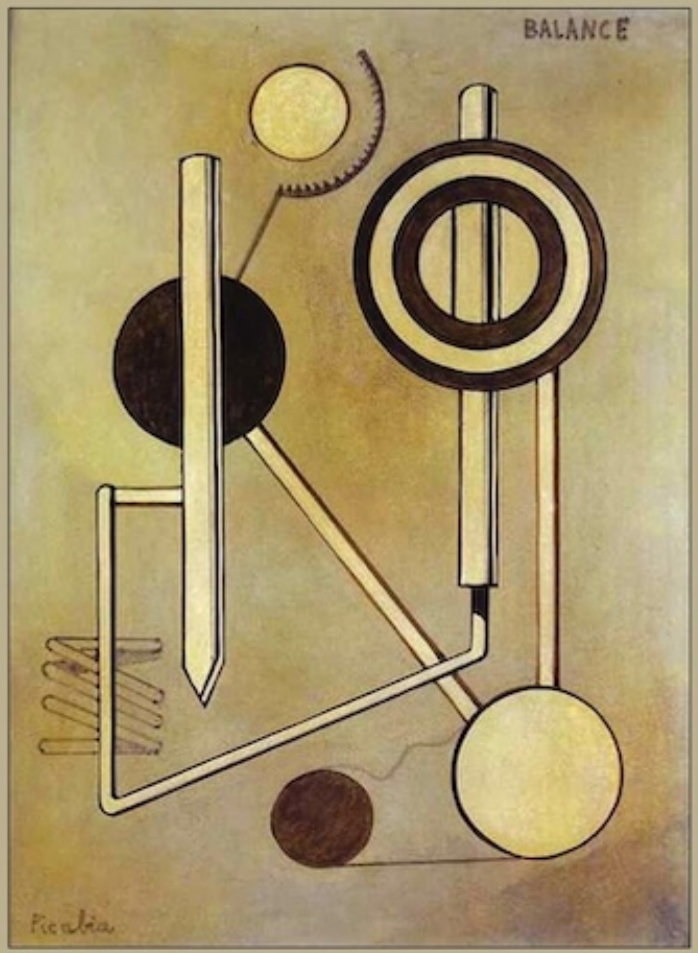
Tools & Process

- Tooling is not on par with “industrial languages”
- Research and industry needs and interests are not always aligned
- FM is a very fragmented landscape with mostly incompatible ecosystems

Conclusion

Philosophical detours




BALANCE
New Edition
AGAINST METHOD
Paul Feyerabend
Introduced by Ian Hacking

Takeaways

Use formal specification to interact with domain experts as early as possible

Model-based Testing is a great way to introduce formal languages and methods

Start small, focusing on important/critical components of the system

Ensure collective code ownership training, pairing, mobbing, mentoring

Select one tool and stick to it (but select wisely)

Do not put proofs on the critical path of software delivery

Santa's List to the FM Community

- Improve tooling and developer experience
- Lower the barrier of entry through more accessible and "practical" training material
- Consolidate the formal languages and methods landscape

Related work

Applying Continuous Formal Methods to Cardano (Experience Report)

James Chapman
Input Output
Glasgow, UK
james.chapman@iohk.io

Arnaud Bailly
Input Output
Nantes, France
arnaud.bailly@iohk.io

Polina Vinogradova
Input Output
Ottawa, Canada
polina.vinogradova@iohk.io

Abstract

Cardano is a Proof-of-Stake cryptocurrency with a market capitalisation in the tens of billions of USD and a daily volume of hundreds of millions of USD. In this paper we reflect on applying formal methods, functional architecture and Haskell to building Cardano. We describe our strategy, projects, lessons learned, the challenges we face, and how we propose to meet them.

CCS Concepts: • **Software and its engineering → Formal software verification**.

Keywords: Agda, Formal Methods, Software Engineering, Cardano, Distributed systems verification

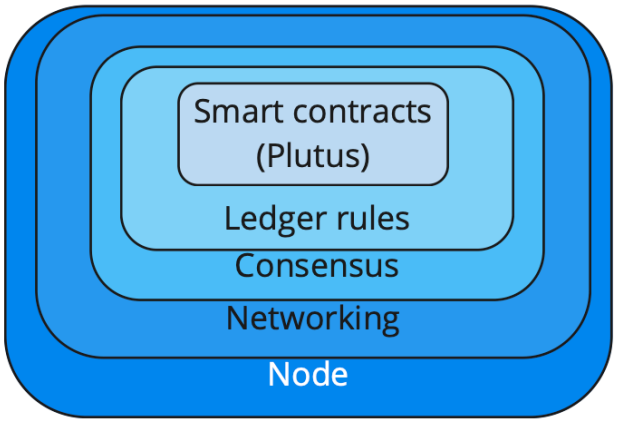


Figure 1. Cardano node layers

Figure 10: FUNARCH'2024

Peras website and code repository contain details about the project

Thanks

- My colleagues at IOG from whom I learnt a lot
- BOBKonf organisers for inviting me
- Josselin Auguste, Bertrand Bougon, Emmanuel Gaillot, Pascal Grange, Fabien Lamarque, Xavier Maso, Matthias Neubauer, and Hugo Traverson for improving it
- Christophe Thibaut for the inspiration
- **You**

Questions?