

Programs without bugs

How we can write programs without any bugs?

Kristijan Šarić, [EXACT BYTE](#)



Who am I?

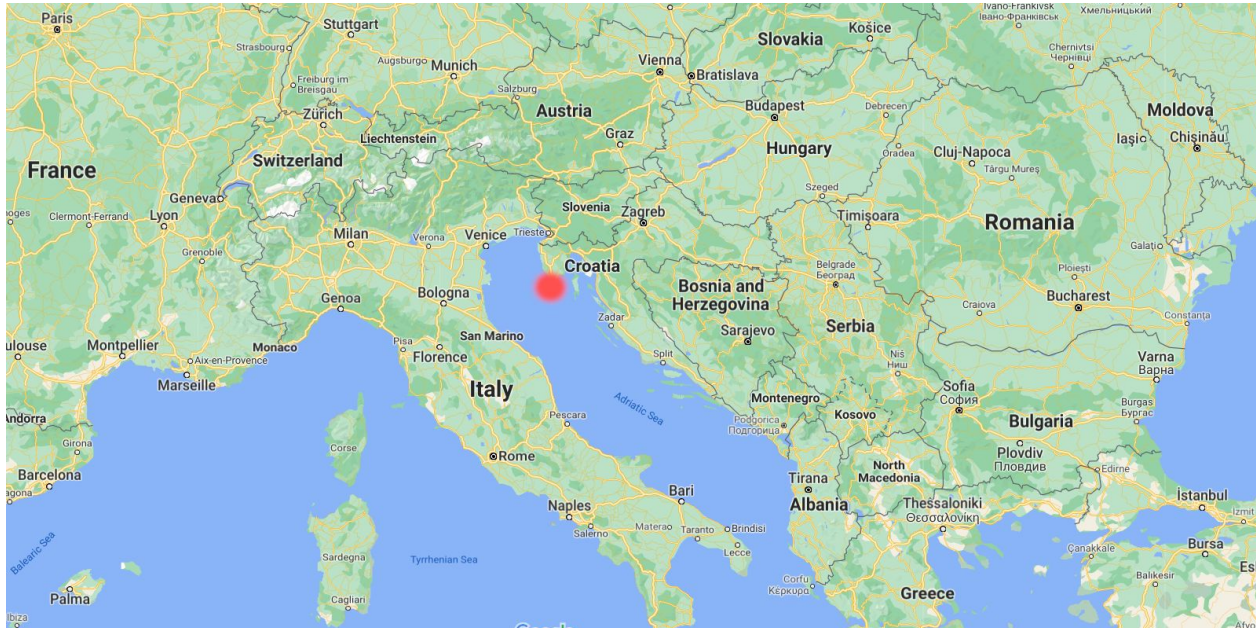
- 15+ years of experience with programming (Java, Haskell, ...) and the last 5+ years experience with AI/ML
- Products using NLP (entity recognition, sentiment analysis):
 - <https://www.emprovio.com/>
 - <https://alenn.ai/>
 - <https://contetino.com/>
- [Mobile application for recognizing the sign language alphabet](#) (for “deaf” people)
- [Application for recognizing breast cancer](#)
- Application for chest radiograph diagnosis (CheXpert, in progress, cooperation with a doctor)
- Non-medical:
 - Web application for detecting parking spaces, working on a small device (think Raspberry Pi)
 - Application for recognizing roads and road signs, "Mini Tesla" project on a small car
 - Application for automatic fault detection in automation
- <https://exact-byte.com/en-blog/>
- Haskell developer, Haskell squad lead at IOHK where a lot of (in)formal proof were done, not part of the formal specification group

Why this presentation?

- A series of **free** applications/articles to promote myself and my company
- An opportunity to open up doors with somebody interested to cooperate

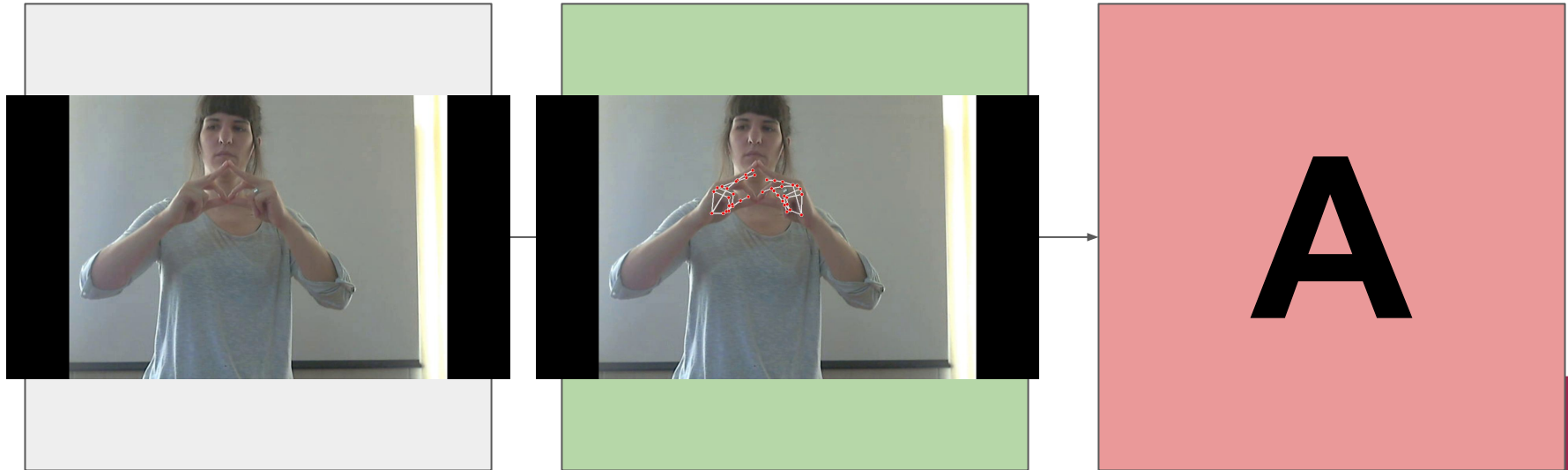
Where do I live?

- Pula, city in Croatia, on the coast, near Italy - <https://en.wikipedia.org/wiki/Pula>



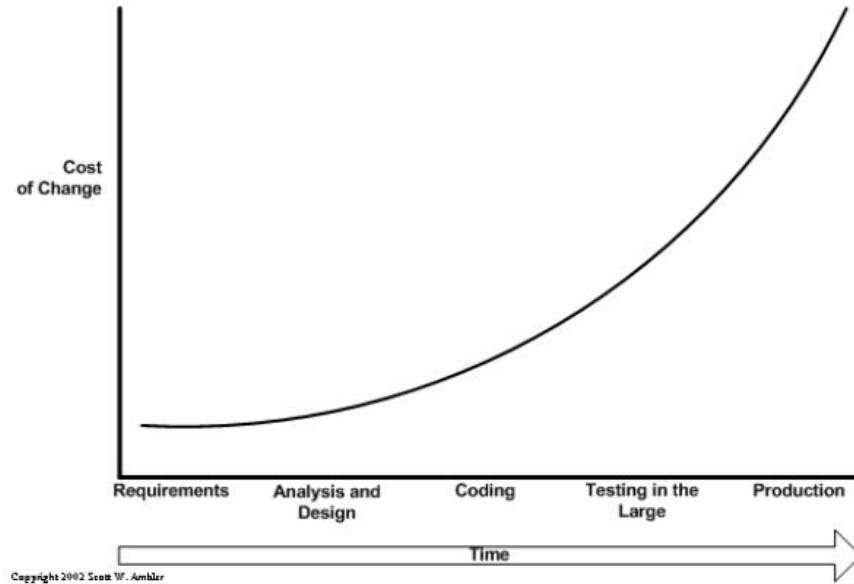
Mobile application for recognizing the sign language alphabet (for deaf people)

<https://youtu.be/7fXDFWrAA6Q>



Correct programs

Slika 1 Traditional cost of change curve



Izvor: Ambler, S. (2014): Examining the Agile Cost of Change Curve,
<http://www.agilemodeling.com/essays/costOfChange.htm> (15. lipanj 2015.)

How do we know our programs work correct(ly)?

1. Correct?



How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.



How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.



How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.
4. We run it with a single input, it works with that, therefore, it **must** be correct.



How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.
4. We run it with a single input, it works with that, therefore, it **must** be correct.
5. We write a test that covers a single input, **guaranteed** it works correctly.



How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.
4. We run it with a single input, it works with that, therefore, it **must** be correct.
5. We write a test that covers a single input, **guaranteed** it works correctly.
6. We write a test that covers two inputs, **absolutely** works correctly.



How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.
4. We run it with a single input, it works with that, therefore, it **must** be correct.
5. We write a test that covers a single input, **guaranteed** it works correctly.
6. We write a test that covers two inputs, **absolutely** works correctly.
7. We write a test that covers multiple inputs and some border cases, this is the **absolute truth**.




How do we know our programs work correct(ly)?

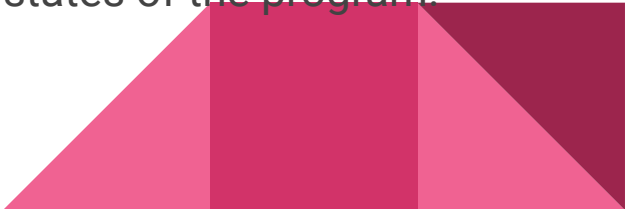
1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.
4. We run it with a single input, it works with that, therefore, it **must** be correct.
5. We write a test that covers a single input, **guaranteed** it works correctly.
6. We write a test that covers two inputs, **absolutely** works correctly.
7. We write a test that covers multiple inputs and some border cases, this is the **absolute truth**.
8. **THE ABYSS**



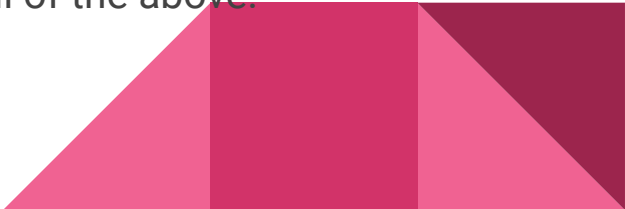
How do we know our programs work correct(ly)?

1. Correct?
 2. I don't have the time for it now.
 3. We close our eyes and wish really hard that it actually works.
 4. We run it with a single input, it works with that, therefore, it **must** be correct.
 5. We write a test that covers a single input, **guaranteed** it works correctly.
 6. We write a test that covers two inputs, **absolutely** works correctly.
 7. We write a test that covers multiple inputs and some border cases, this is the **absolute truth**.
 8. **THE ABYSS**
 9. We have tests that are auto-generated and check if our **invariants** hold.
- 

How do we know our programs work correct(ly)?

1. Correct?
 2. I don't have the time for it now.
 3. We close our eyes and wish really hard that it actually works.
 4. We run it with a single input, it works with that, therefore, it **must** be correct.
 5. We write a test that covers a single input, **guaranteed** it works correctly.
 6. We write a test that covers two inputs, **absolutely** works correctly.
 7. We write a test that covers multiple inputs and some border cases, this is the **absolute truth**.
 8. **THE ABYSS**
 9. We have tests that are auto-generated and check if our **invariants** hold.
 10. We have state machine tests that cover all the important states of the program.
- 

How do we know our programs work correct(ly)?

1. Correct?
 2. I don't have the time for it now.
 3. We close our eyes and wish really hard that it actually works.
 4. We run it with a single input, it works with that, therefore, it **must** be correct.
 5. We write a test that covers a single input, **guaranteed** it works correctly.
 6. We write a test that covers two inputs, **absolutely** works correctly.
 7. We write a test that covers multiple inputs and some border cases, this is the **absolute truth**.
 8. **THE ABYSS**
 9. We have tests that are auto-generated and check if our **invariants** hold.
 10. We have state machine tests that cover all the important states of the program.
 11. We have a formal specification of the program that includes all of the above.
- 

How do we know our programs work correct(ly)?

1. Correct?
2. I don't have the time for it now.
3. We close our eyes and wish really hard that it actually works.
4. We run it with a single input, it works with that, therefore, it **must** be correct.
5. We write a test that covers a single input, **guaranteed** it works correctly.
6. We write a test that covers two inputs, **absolutely** works correctly.
7. We write a test that covers multiple inputs and some border cases, this is the **absolute truth**.
8. **THE ABYSS**
9. We have tests that are auto-generated and check if our **invariants** hold.
10. We have state machine tests that cover all the important states of the program.
11. We have a formal specification of the program that includes all of the above.
12. We have a formal verification of the program that covers 100% of all cases (**used only in critical applications**) - the specification adheres to the program, they are equal

Formal specification

- https://en.m.wikipedia.org/wiki/Formal_specification



Formal specification

- https://en.m.wikipedia.org/wiki/Formal_specification
- In computer science, formal specifications are mathematically based techniques whose purpose are to help with the implementation of systems and software. **They are used to describe a system, to analyze its behavior, and to aid in its design by verifying key properties of interest through rigorous and effective reasoning tools.**[1][2] These specifications are formal in the sense that they have a syntax, their semantics fall within one domain, and they are able to be used to infer useful information.[3]



Formal specification

- Formal (form, shape)
- Specification (specifying something)
- Let's take an example!



Example

Missing example/slides about Coq and formal proofs. It would make things more confusing.



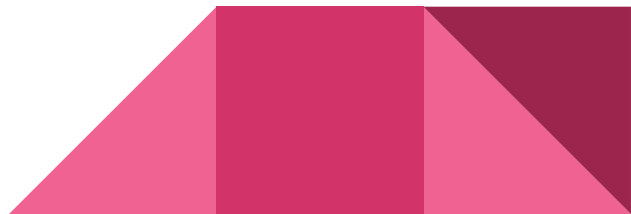
Example

- Why Haskell?



Example

- Why Haskell?
- An interesting programming language where a lot of ideas (research) is “put into production” (tested)



Example

- Why Haskell?
- An interesting programming language where a lot of ideas (research) is “put into production” (tested)
- The story about dependently typed Haskell is a loong one...



Example

- Why Haskell?
- An interesting programming language where a lot of ideas (research) is “put into production” (tested)
- The story about dependently typed Haskell is a loong one...
- **Pure** functional language

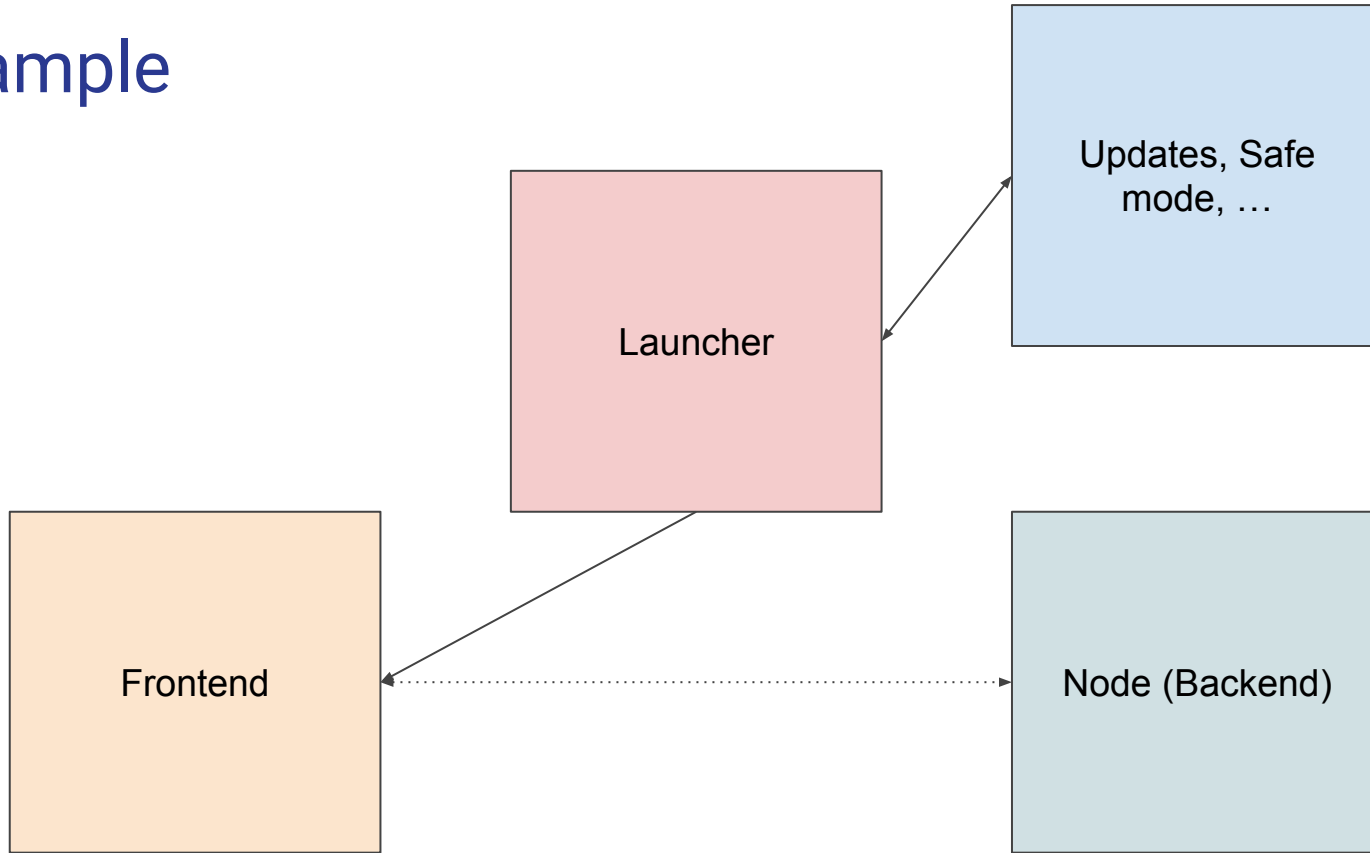


Example

- <https://github.com/input-output-hk/cardano-shell>
- Launcher, the program that starts up the crypto “wallet” frontend and the node backend and worries about updates
- There were tons of bugs with the existing program, there are a lot of details that can mess up the actual flow of the program
- After I made the specification (for the possible states and how the program behaves in them) I added the tests that would make sure that the program does exactly that
- <https://github.com/input-output-hk/cardano-shell/tree/master/cardano-launcher>

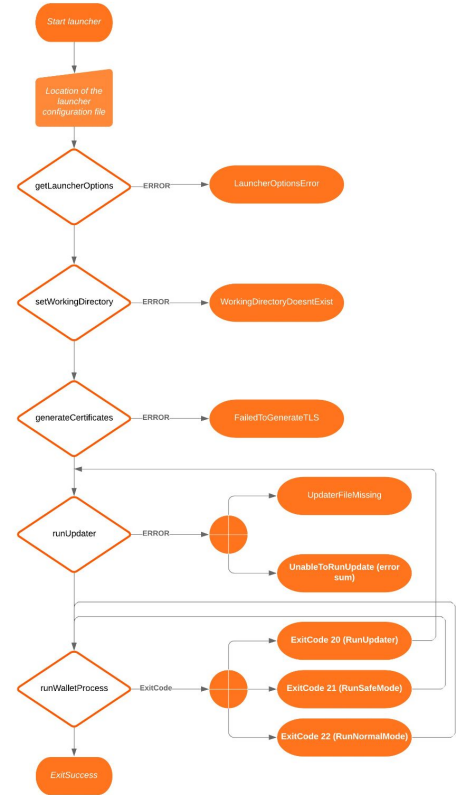


Example



Example

- <https://github.com/input-output-hk/cardano-shell/tree/master/cardano-launcher>



Model

- The (simplest) **pure** representation of the actual program
- You can reiterate and add complexity
- Pure, simplified, is math - in this case let's limit ourselves to mapping:
 - IF True THEN 'A' ELSE 'B'
 - { True => 'A', False => 'B' }
- We have lists, structures, data, but only pure values and mapping
- What would be the simplest model for our problem?



(Finite) State Machine

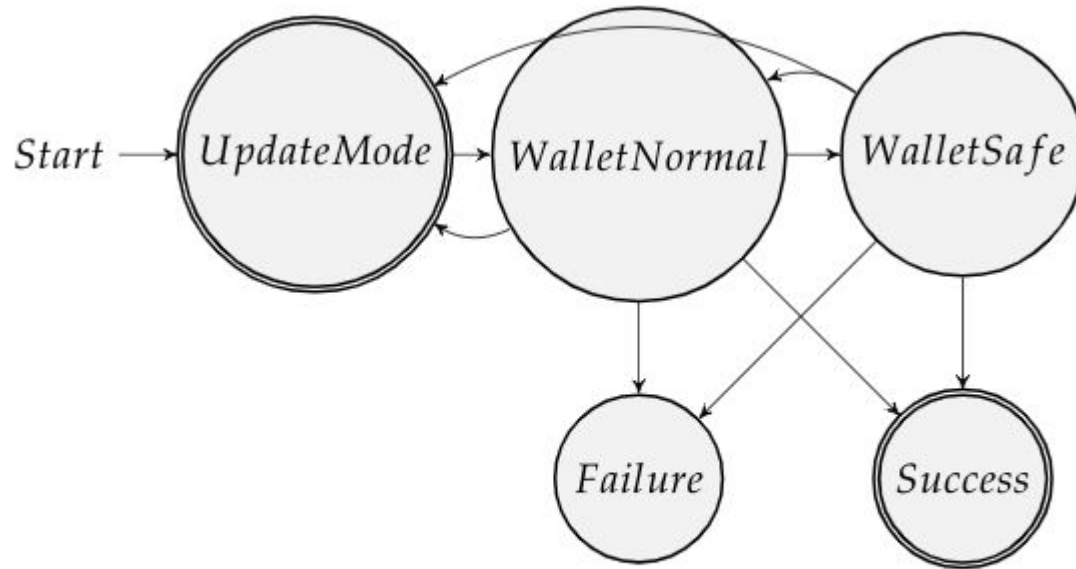


Figure 8: Update FSM.

TLA+

- https://en.wikipedia.org/wiki/Modal_logic
- Language for specifying programs
- There are limits, for those who are interested doing something more rigorous try Isabelle or Coq
- https://en.wikipedia.org/wiki/Model_checking
- <https://github.com/input-output-hk/cardano-shell/blob/master/cardano-shell/specs/tla/UpdateSystemWallet.tla>
- <https://github.com/input-output-hk/cardano-shell/blob/master/cardano-launcher/test/LauncherSMSpec.hs>
-



That's all folks

- It's useful to know what your program needs to do before coding
- Specification is very useful when used in critical parts of the code
- When you think about not only your program, but about how to prove your program correct, you will understand more about your code than before
- Thanks for your time
- Resources:
 - <https://softwarefoundations.cis.upenn.edu/>
 - “FORMALNI DOKAZI U PROGRAMIRANJU”, Kristijan Šarić, 2016.
- Questions?

