



axini

---



MDD/MBT at ProRail (ERTMS)

Axini Model Based Testing

---

**axini**

---

**MDD/MBT at ProRail (ERTMS)**

**Welcome, nice to meet you!**

---



---

# In a nutshell

- A complex system in a complex environment for which ProRail applied MDD with MBT from the start of the project.
- ProRail saved (at least) 5.000 testing hours (on a project of 20.000 hour).
- The project was ready half a year before deadline (for a 2 year project).
- ProRail decided to use SAFe (this was the first project).
- And of course there was Covid.

---

# Who am I?

---

axini



- Introduction ProRail and Axini
  - The ETIS system (part of ERTMS)
  - Why are systems like ETIS hard?
  - The Axini Modeling Platform
  - Lessons learned
-

---

# ProRail

ProRail is a Dutch government organization responsible for

- the maintenance and extension of the national railway network infrastructure (not the metro or tram),
- the allocation of rail capacity, and
- controlling rail traffic.

---

# ProRail in numbers

- Every day 5.500 trains on the 7.000 kilometer (4.350 mile) long Dutch rail track
- Assets on the tracks
  - 11.578 signals,
  - 6.256 switches,
  - 2.393 grade crossings,
  - 398 stations,
  - 68 movable bridges,
  - 26 tunnels.
- Yearly 22.000 issues and 4.654 calamities.
- 2022 budget: €892M

---

# ProRail, the biggest cyber physical system in the Netherlands?





---

# Incidents

- <https://www.youtube.com/watch?v=cclVfWaKPyQ>



# ERTMS technology

## The technology behind ERTMS

**ERTMS Level 2**  
**Yellow trackside balises** on sleepers, in place of signals

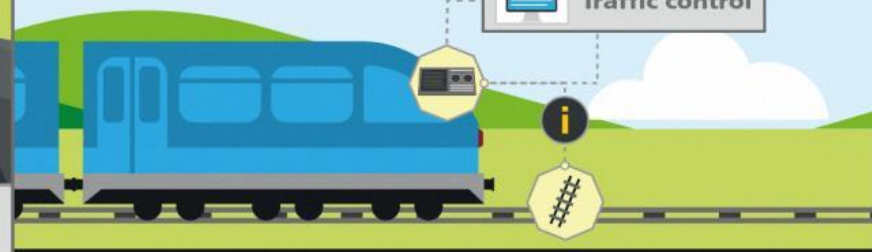


From now on, the **driver** will watch his display screen in the train to see what he is able to and permitted to do



**GSM-R** as the communication tool between track and train at both Level 2 and 3

### ERTMS Level 3



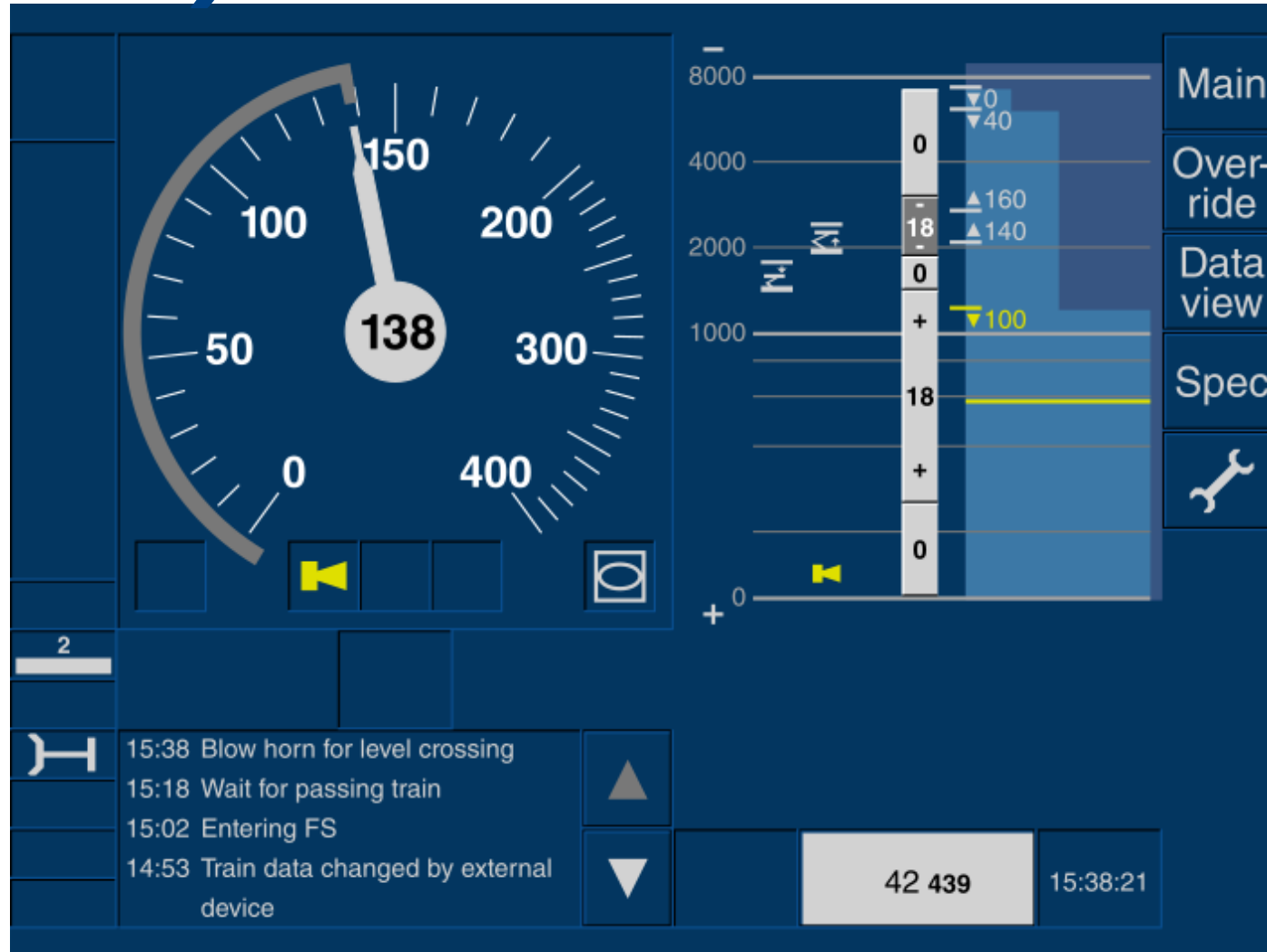
**Train detection** will disappear from the infrastructure

**Phased and controlled**



from **analogue** to **digital**

# ERTMS Technology (Driver Machine Interface)

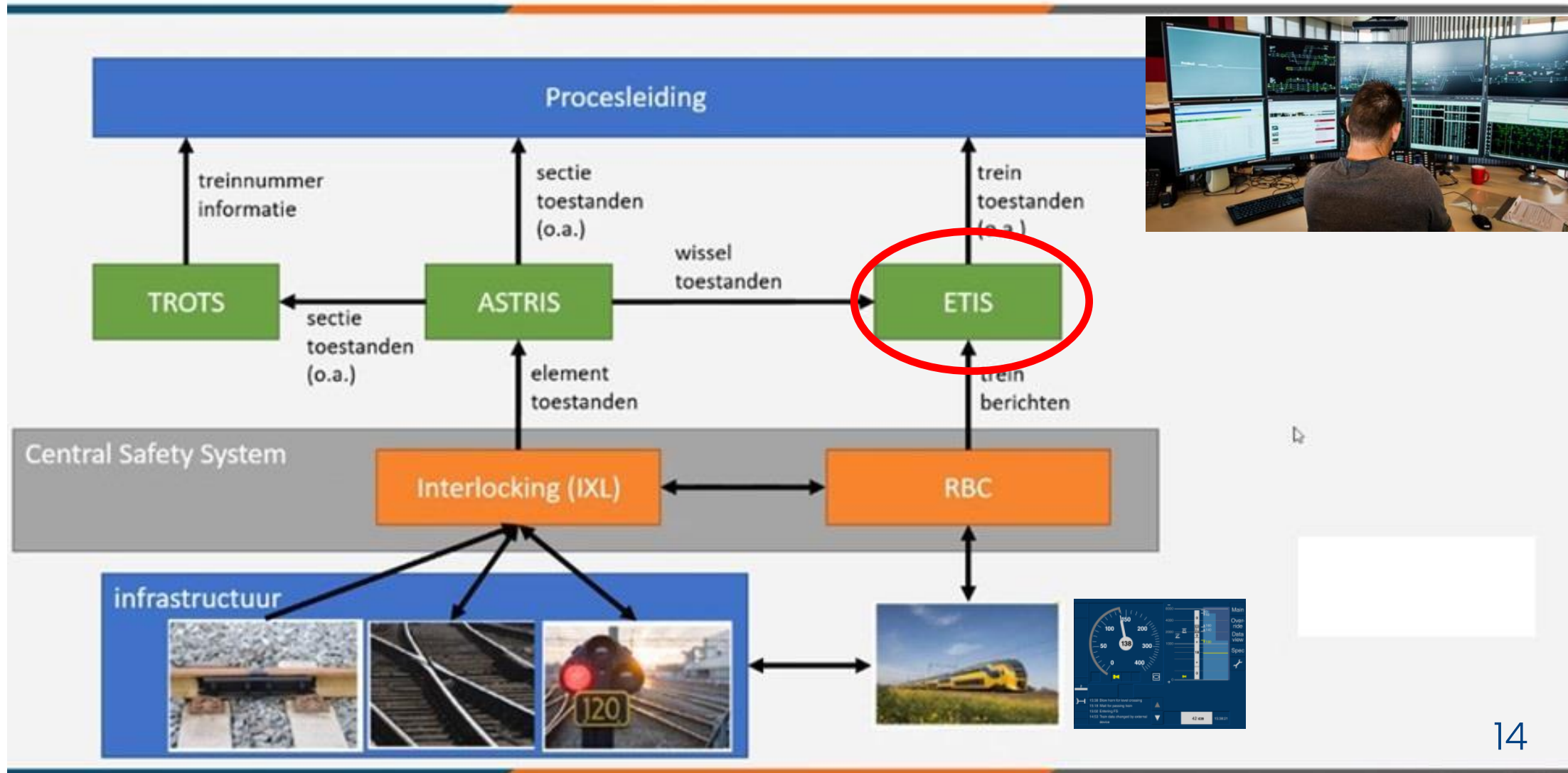


---

# ERTMS technology (traffic control)



# ETIS (ERTMS train information system)



**axini**

---



How to test such a system?

---



axini

---

What would you like to hear?

---

---

# Some options

- More about Axini MBT and the Axini Modeling Platform
- More about why it's so difficult to test systems like ETIS
- More about the experience and lessons learned



**axini**

---

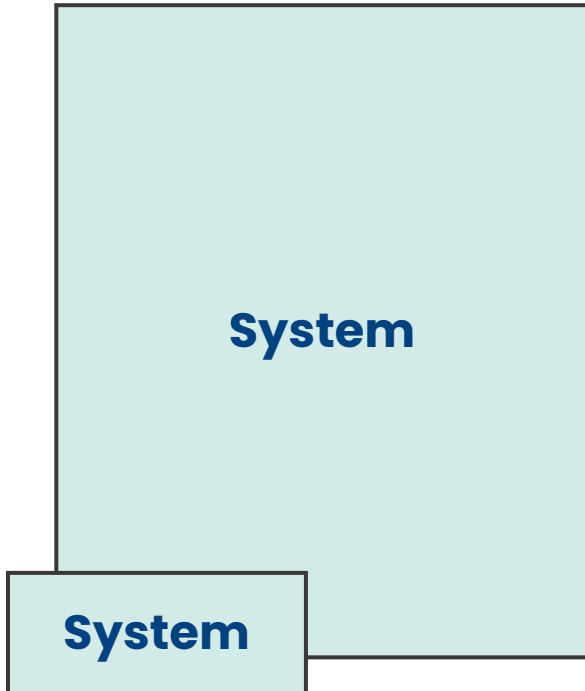
**How to test complex systems?**

**How to make reliable software?**

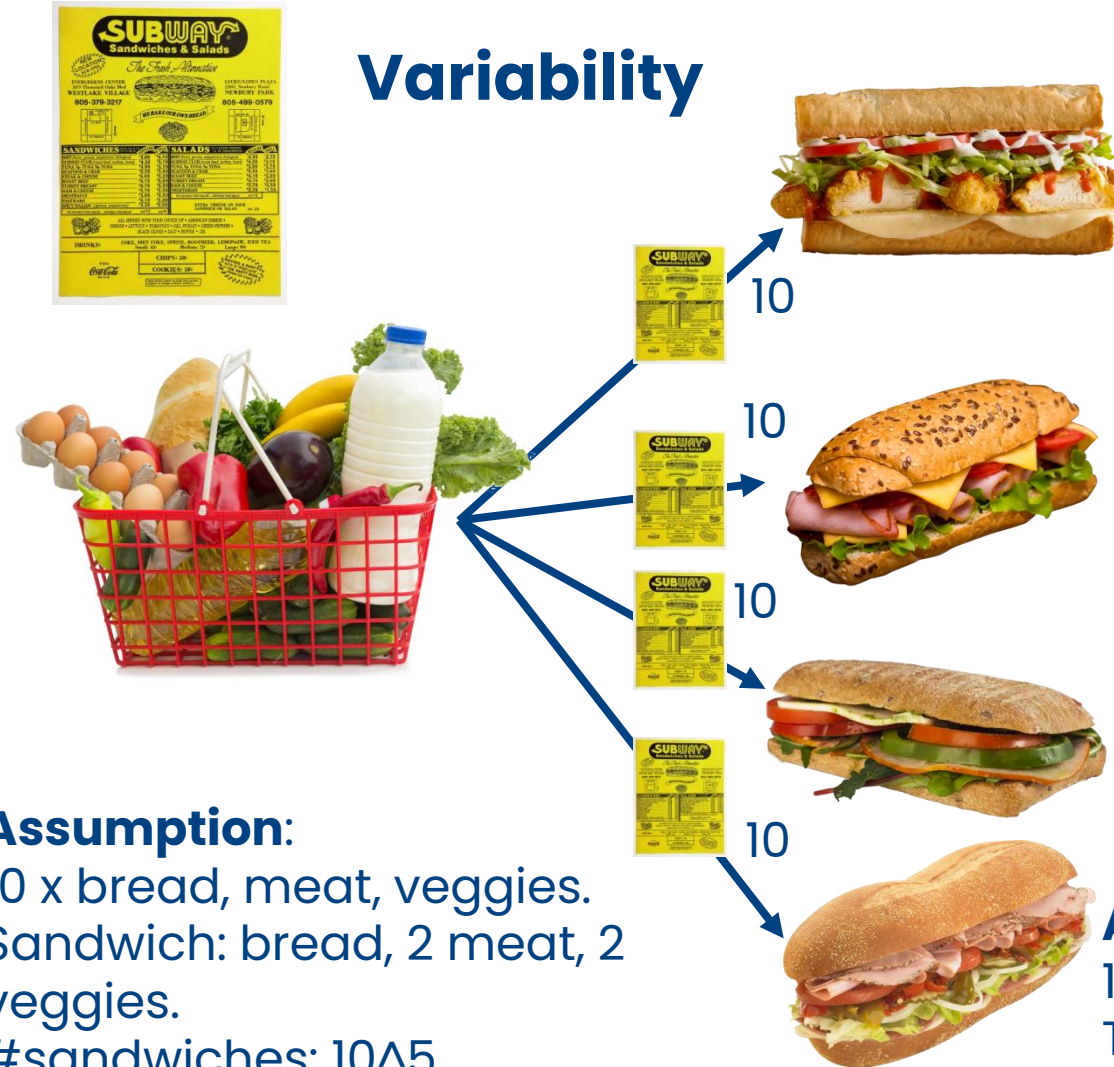
---

# Challenges for complex systems

## Size

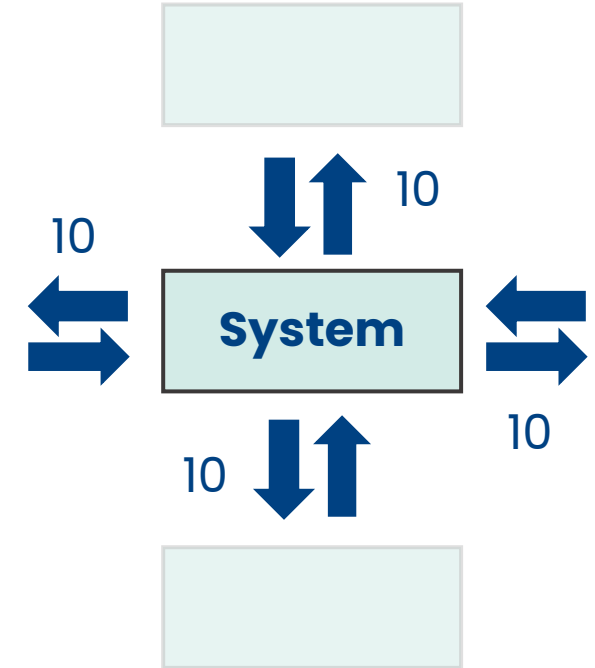


## Variability



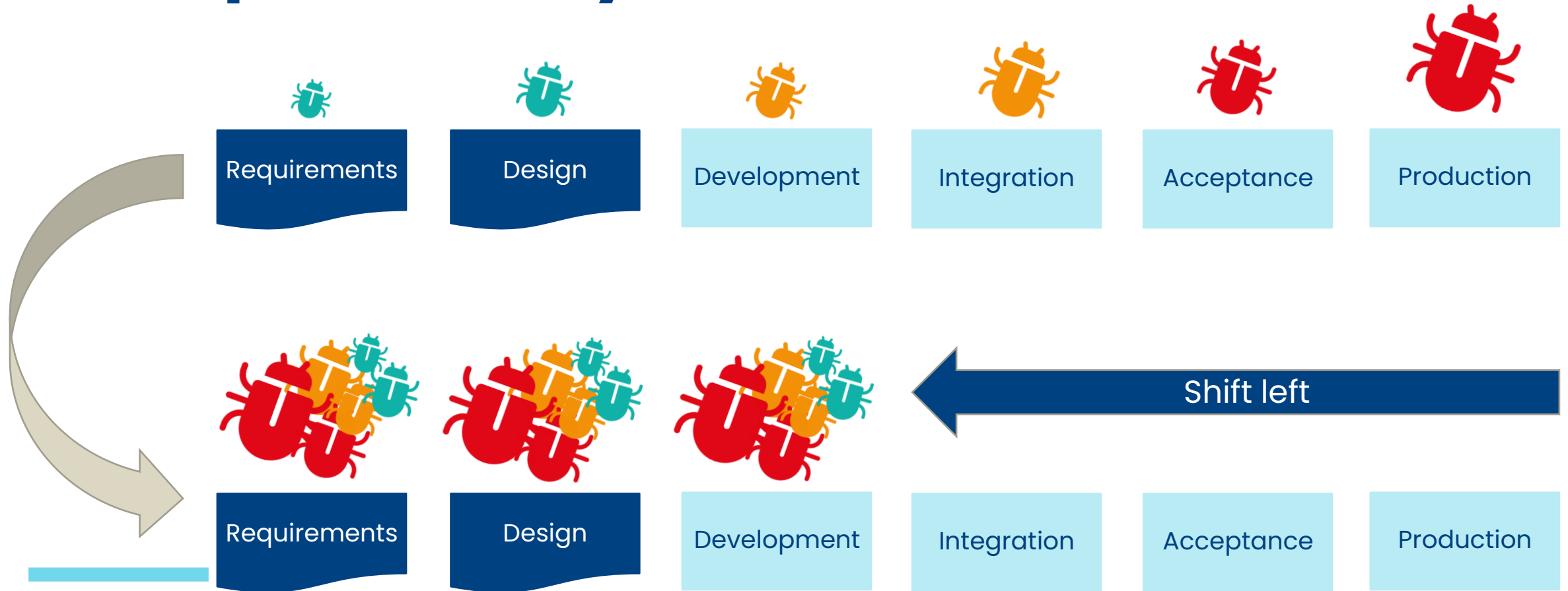
**Assumption:**  
 10 x bread, meat, veggies.  
 Sandwich: bread, 2 meat, 2  
 veggies.  
 #sandwiches:  $10^5$

## Connectivity

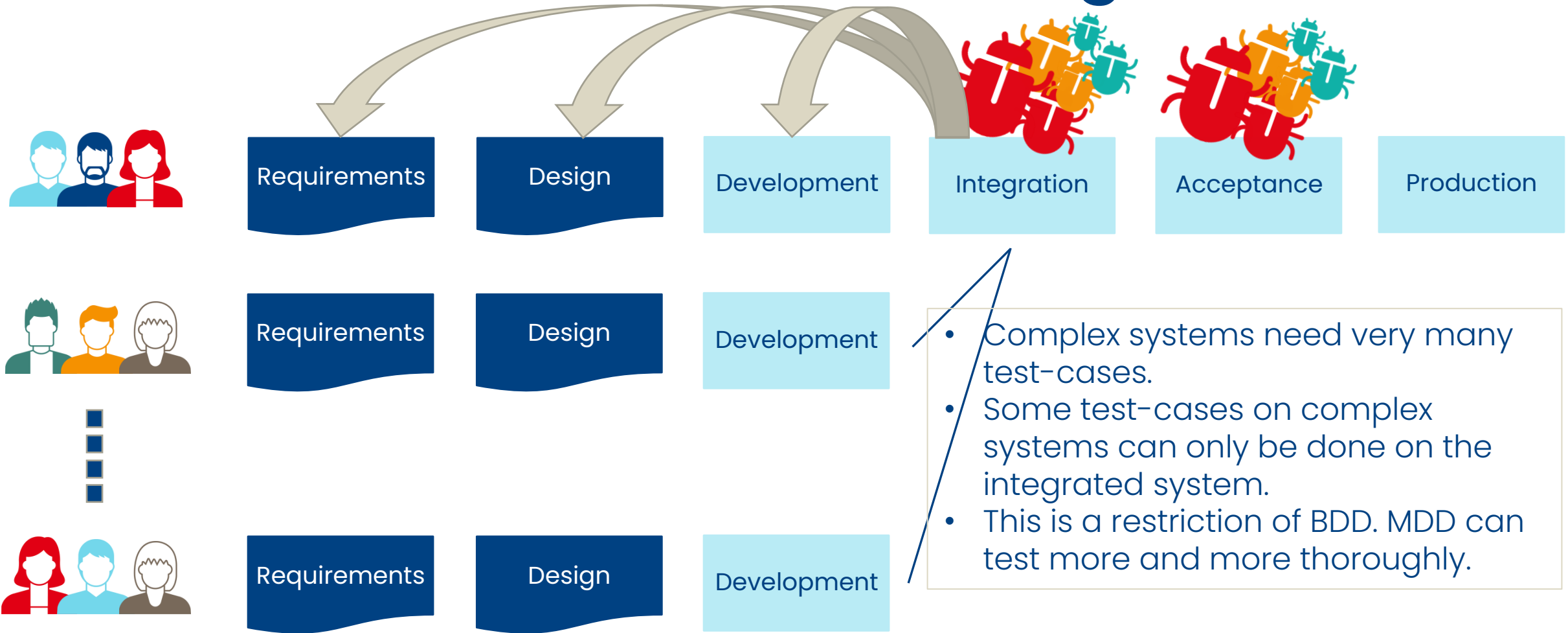


**Assumption:**  
 10 interactions per Interface  
 Total # interactions  $10^4$

# Boehm's law: The cost of bugs grows exponentially with time



# Problems come at/after integration
















---

# Types of test automation

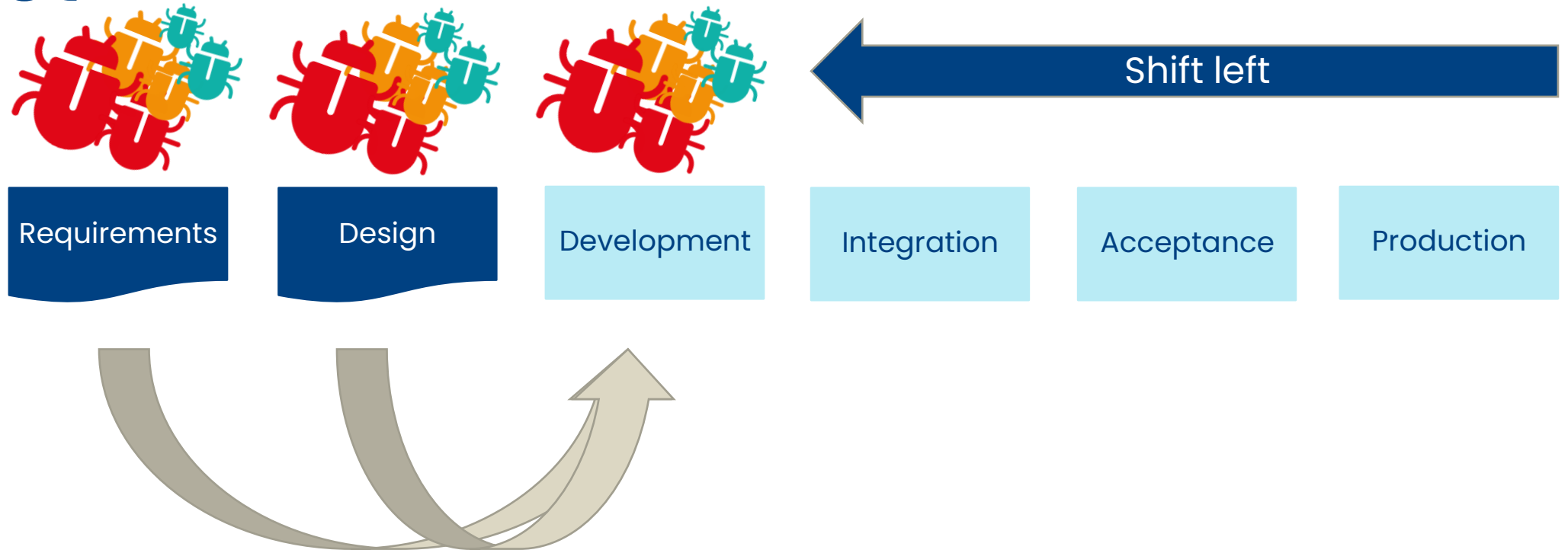
- Scripting: automates test-execution
  - BDD (Behavior Driven Development)
  - TDD (Test Driven Development)
  - Unit-tests
  - ...
- Model Based Testing: automates the entire test-process.
  - MDD (Model Driven Development)
- Only BDD and MDD relate requirements and testing

# Test tool comparison

	Process	Hand	BDD
Design	Make specification		
	Make model		
Test	Make test		
	Predict outcome		
	Script test		
	Execute test		
	Evaluate outcome		

 Manual step  
 Automated

# MDD: couple requirements, design and test



**axini**

---

# Axini MDD/MBT in a nutshell



---

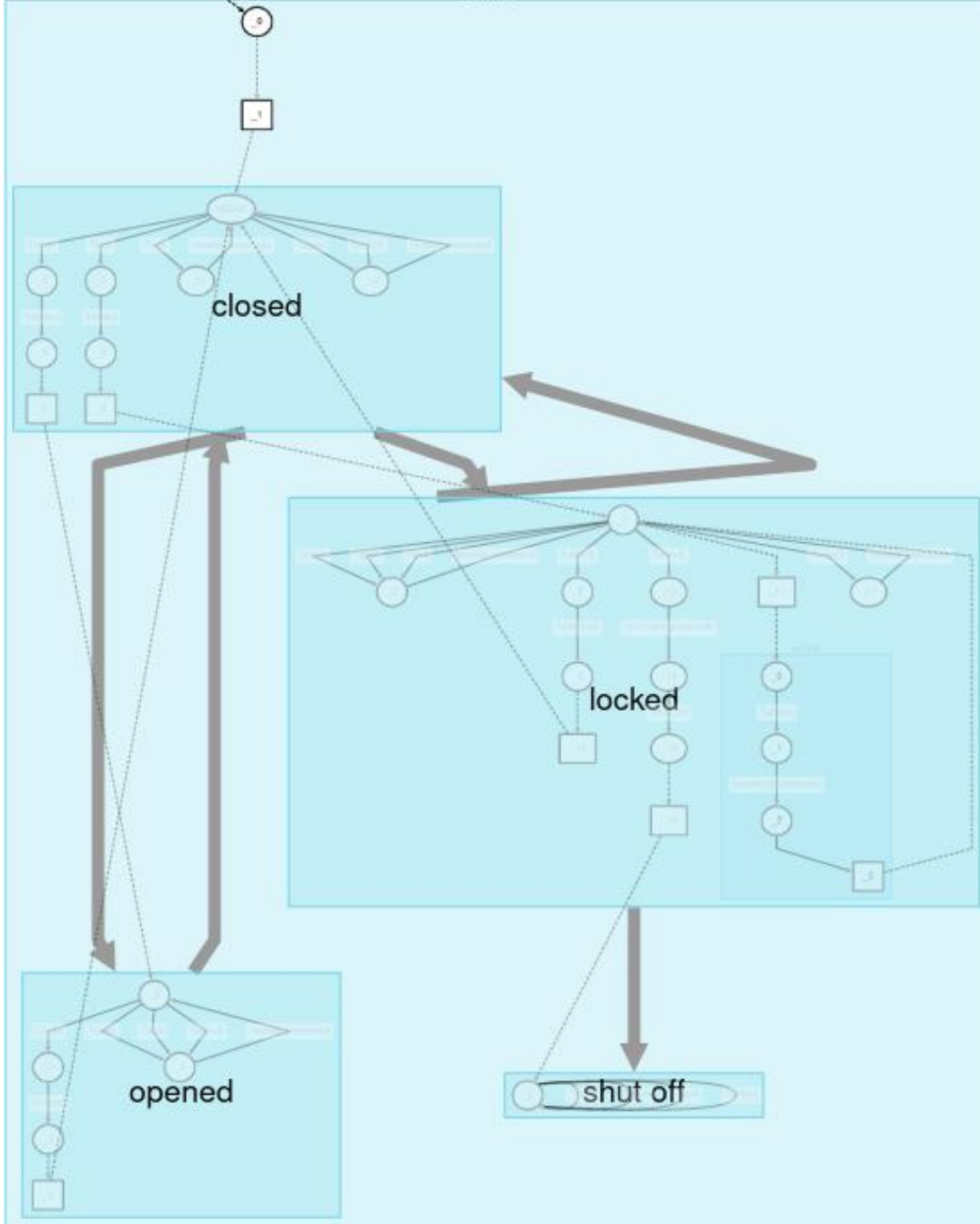
# Some examples in High-Tech

- Interface modeling and testing
- Complex business logic modeling and testing
- Systems modeling and testing
  
- Thermo Fisher Scientific
- ProRail
- ITAB

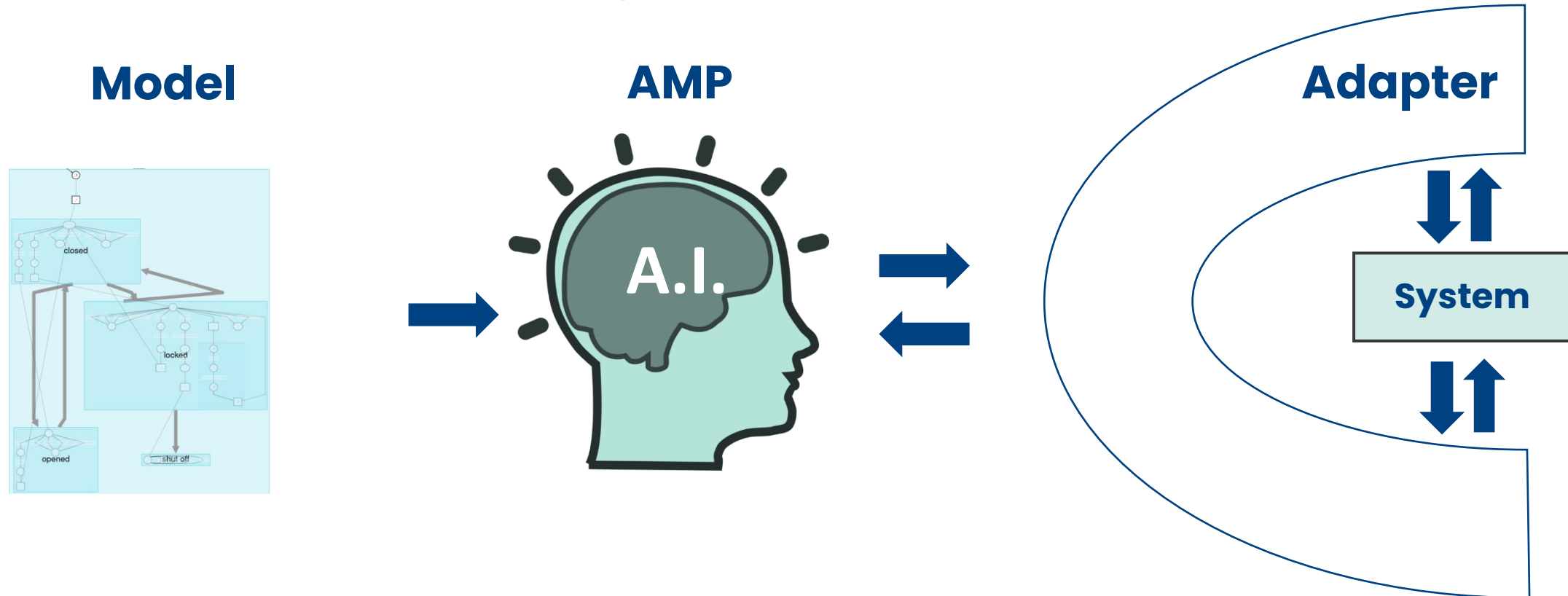
---

# Some examples in Finance

- Example clients: Achmea, Campina Pension Fund, Robeco Investment Bank, Top 3 Bank NL
- Business rules and calculation rules
  - Pension calculations,
  - URM,
  - Disbursement (Dutch: Excasso),
  - Investment portfolio optimization,
  - Life and non-life insurances,
- Client communication (letters, emails, etc.)
- Online transactions and batches pension administration.
- Pension administration (rest-portfolios).



# Axini Modeling Platform (AMP)



Axini automates the entire test process based on the specification/model.

- Automated test-case generation (including test-data).
- Automated test-case execution.
- Automated test-case evaluation.

# Axini MBT TomTom analogy





The crux of the Axini MBT solution is the model. One can compare this approach with a TomTom.

- A TomTom does not explicitly keep track of routes, but derives these from a map, viz. model. In a comparable way, Axini generates test-cases from the model.
- Just as a TomTom can dynamically change routes, Axini can dynamically derive test-cases. For example for good weather/bad weather, to zoom in on changes/requirements, to work around known errors, etc.
- Just as with a TomTom, with Axini the test-cases are immediately up to date after an update of the model.

# Test tool comparison

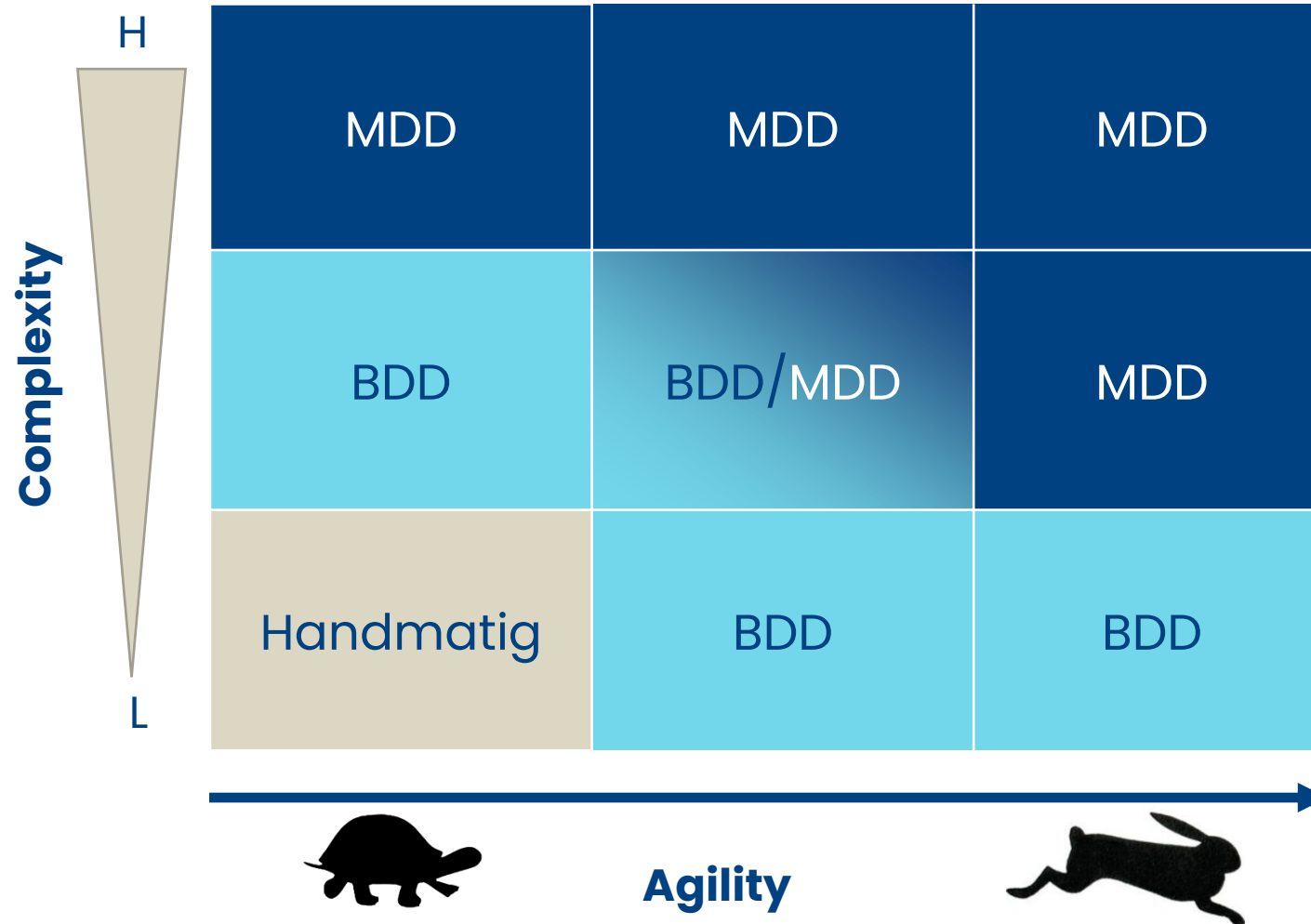
	Process	Hand	BDD	Axini
Design	Make specification	Hand	Hand	Hand, Automated
	Make model			Hand
Test	Make test	Hand	Hand	Automated
	Predict outcome	Hand	Hand	Automated
	Script test		Hand	✗
	Execute test	Hand	Automated	Automated
	Evaluate outcome	Hand	Automated	Automated

 Manual step  
 Automated

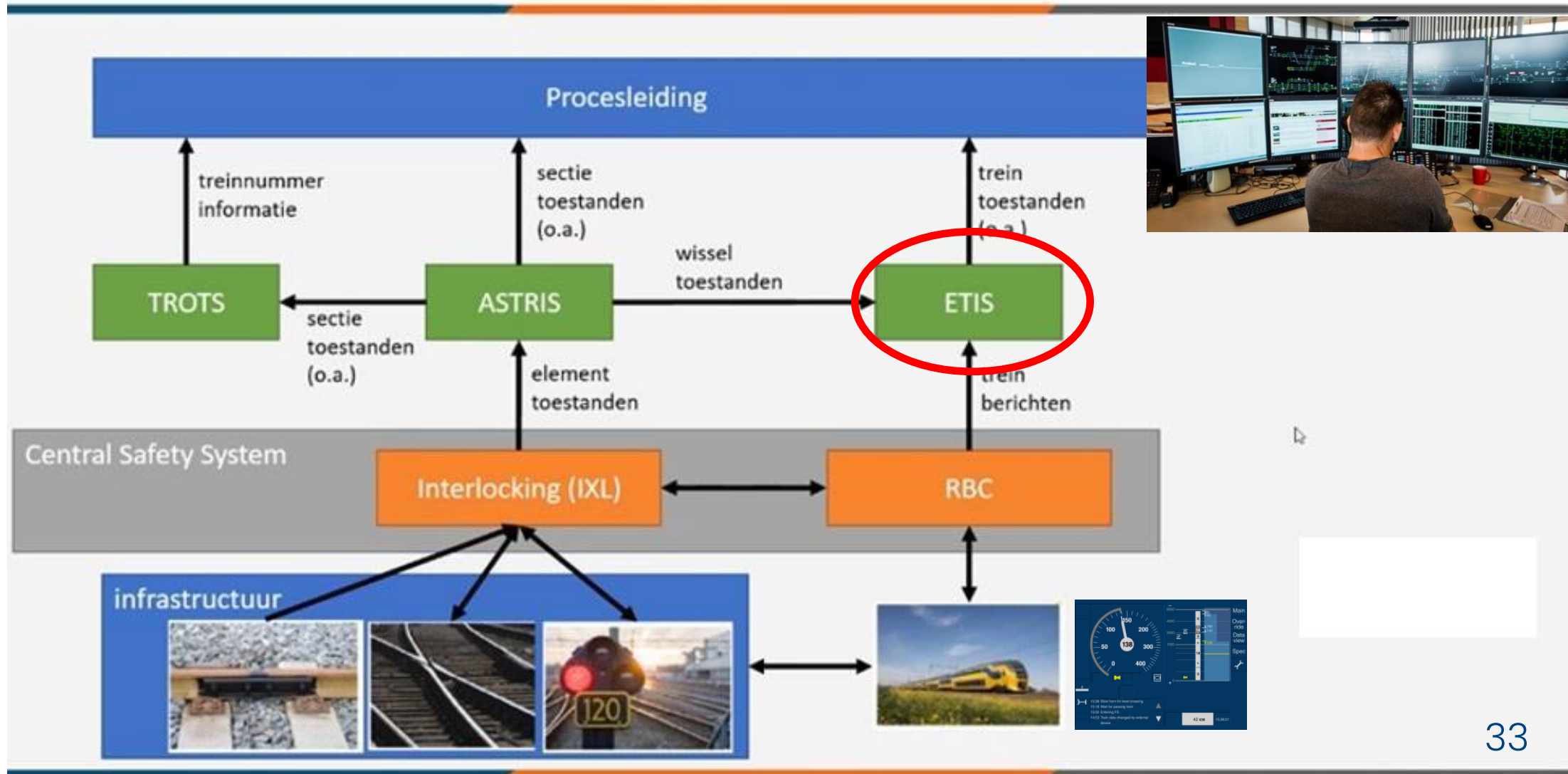
More coverage = More certainty

100% automation

# MDD vs BDD



# Back to ETIS





---

# The ETIS model

- 4 parallel processes
- 10 test-sets
  - Each test-set focusses on a different aspect
- Test-cases with a depth of +4000 steps
- 6 interfaces
  - 4 with big XML XSDs
  - 2 with a big XML XSD, but also encapsulated ETCS messages
  - The XMLs can become rather large

# Lessons learned



---

# It is possible to model systems of the size of ETIS

AML provides the required modeling power

- States and transitions with data (simple and complex types) and time
- Modules, super-states and functions for structuring
- Model configuration and parameterization
- Multiple processes
- Complex data-types

---

# Required features

- Versioned models
- Visualization
- Exploration and debugging
- Scalable, big state-vectors  $O(100+)$
- CI/CD integration
- External git repo integration (e.g., GitLab)
- Standardized and fast adapters

---

# Almost no integration problems

- Smart path coverage testing strategies
- Constraint solvers
- Also the unlikely test-cases are generated
- Many bugs were found during development

---

# MBT and SAFe go together well

- Only unit-tests and MBT.
  - and some manual tests, performance tests, etc
- A passing MBT test is the definition of done.
- Modeling catches errors and ambiguities.
- Modelers can help programmers (and vice versa).
- The whole team should own the models (and be able to model).
- Have a dedicated modeller in your team.

---

# Lessons learned, MBT in practice

- Start modeling **immediately** at the start of the project.
  - This requires input from architects/designers etc.
- Gated MBT in CI/CD.
- MDD/MBT gives project managers control.

---

# Lessons learned

- Start small
- MDD/MBT is a paradigm shift
  - The start is hard
  - It's new for everyone
  - You need information that is not yet there/complete
- Modeling is a real effort
- Modeling (concurrency) is not for everyone.



---

# The numbers

- ProRail saved (at least) 5.000 testing hours (on a project of 20.000 hour).
- The project was ready half a year before deadline (for a 2 year project).
- 0.6 FTE modeler on a 4 FTE team.

---

# But how?

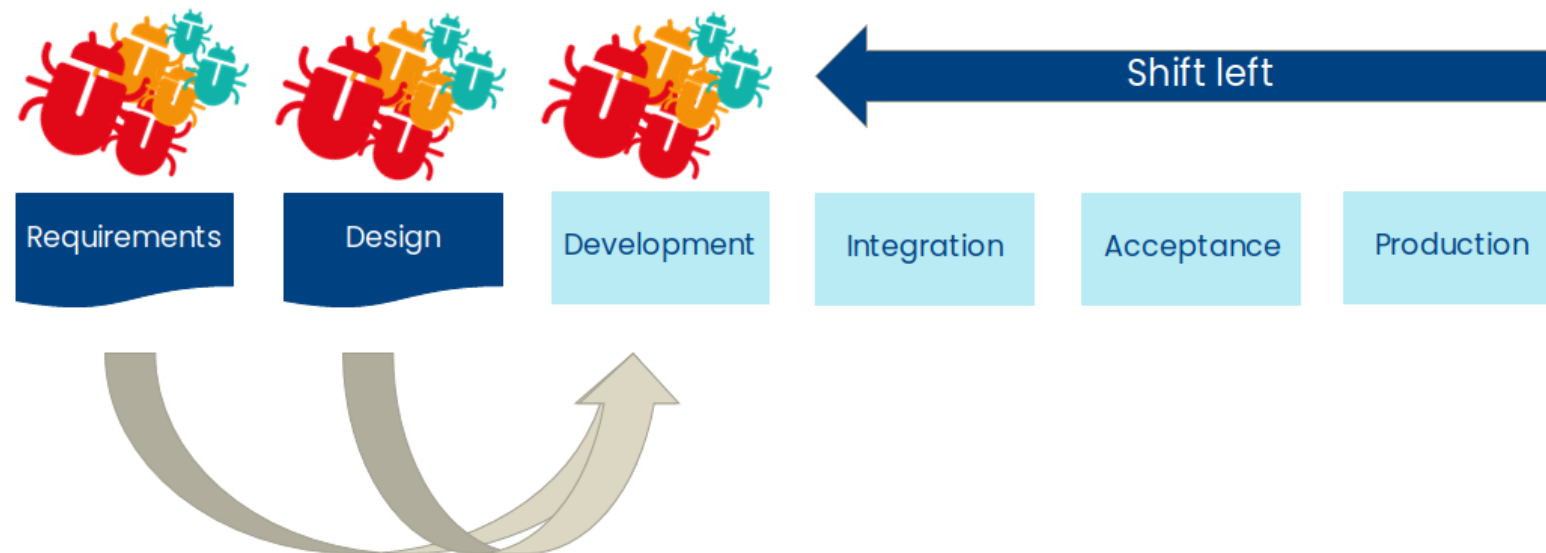
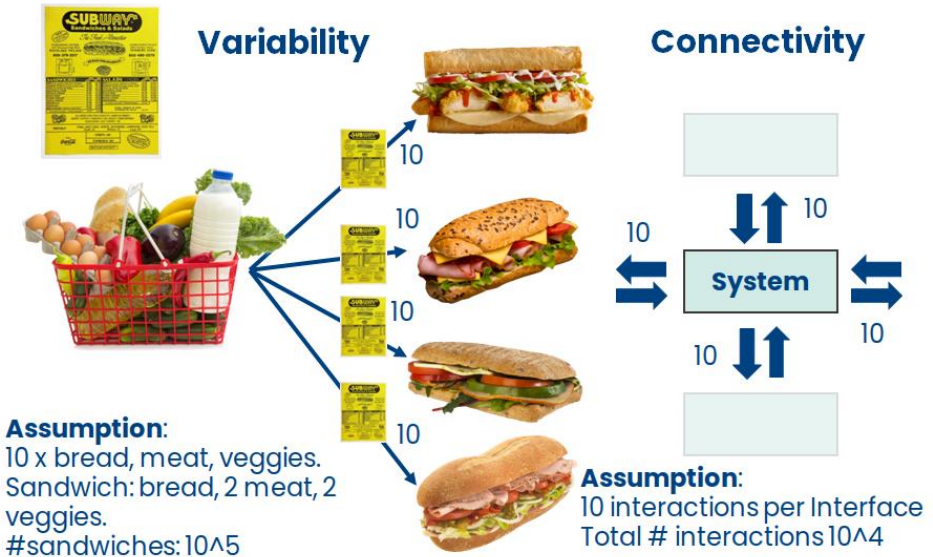
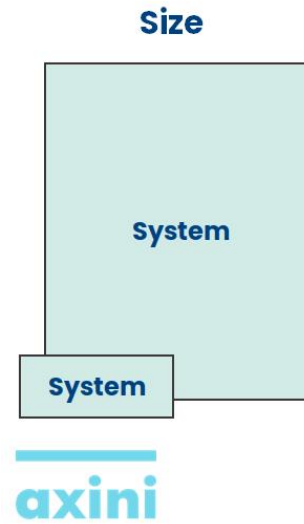
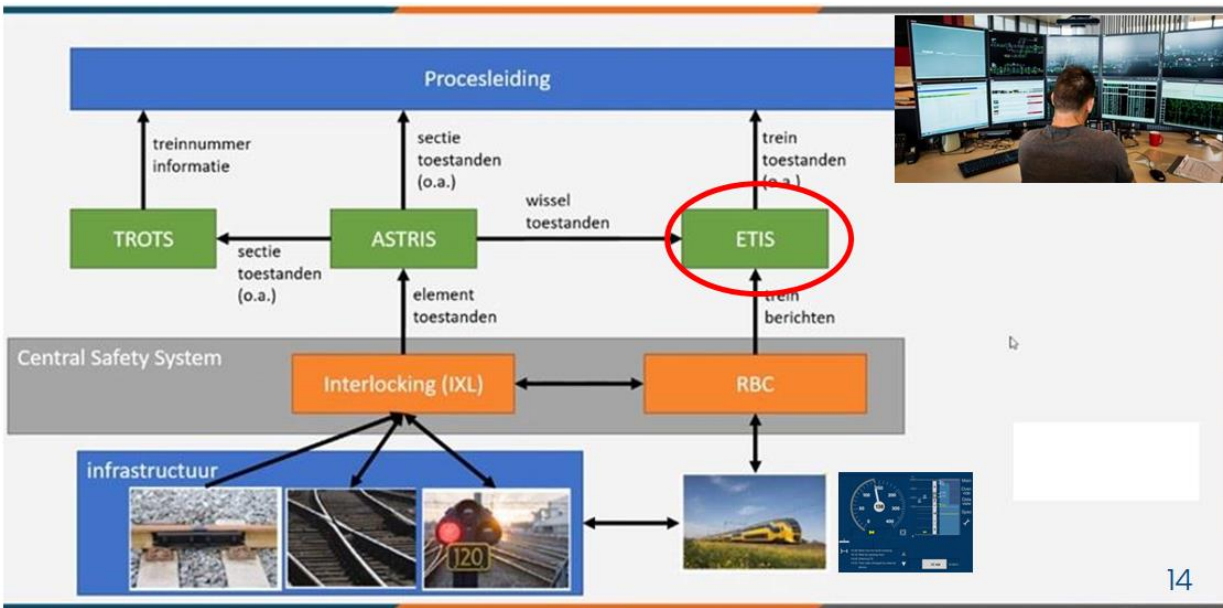
- Join tomorrow's Axini modeling workshop sneak preview!

**axini**

---

**Conclusion**

# Conclusion



Complexity H L	MDD	MDD	MDD
	BDD	BDD/MDD	MDD
	Handmatig	BDD	BDD
		Agility	

The table shows the relationship between Complexity (High to Low) and Agility (Slow to Fast). The bottom row features a turtle icon on the left and a dog icon on the right, representing the spectrum of development approaches.

# Conclusion

## Lessons learned



## In a nutshell

- A complex system in a complex environment for which ProRail applied MDD with MBT from the start of the project.
- ProRail saved (at least) 5.000 testing hours (on a project of 20.000 hour).
- The project was ready half a year before deadline (for a 2 year project).
- ProRail decided to use SAFe (this was the first project).
- And of course there was Covid.

**axini**

---



Questions?

---

# Free 3 hour workshop?

**axini**

---

Contact Machiel van der Bijl  
vdbijl@axini.com  
+31 6 1642 6332

# Thank you!



---

# How to connect?

- Site: `course02.axini.com`
- Password: `testnet22`
- User name: `see paper`



**axini**

---

Model Based Testing with the

**Axini Modeling Platform**

and the infamous

**Coffee Machine**

---

# Axini Modeling Language (AML)

- process + data language
  - inspired by **Promela** (SPIN) and **LOTOS**
- model consist of **parallel processes**
- communication over **hand-shake channels**
  - *external*: communication with SUT
  - *internal*: communication between processes

*Largest AML  
model: >10k loc.*

## *Semantics*

*A process in AML is  
mapped upon a  
(symbolic) labelled  
transition system.*

## **behavioral part:**

- **stimuli** (inputs)
- **responses** (outputs)
- (non-deterministic) **choice**
- **repeat**
- states / **goto**

*only needed  
today*

## **data part:**

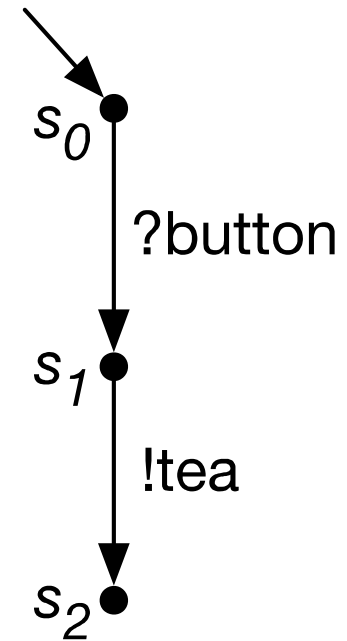
- Ruby-like, **strictly typed**
- messages can be **received** and **send**
  - **label** (name)
  - **parameters** (attributes)
- process can have **variables**

*AML is implemented as a **Ruby DSL**: Ruby can be used as preprocessor.*

# Hello, Tea Machine

```
external 'extern' name of the external channel
process('button-tea') {
  # declarations of labels, variables
  timeout 10.0 timeout for all responses
  stimulus 'button', on: 'extern'
  response 'tea', on: 'extern'

  # behaviour of the process
  receive 'button'
  send 'tea'
}
```

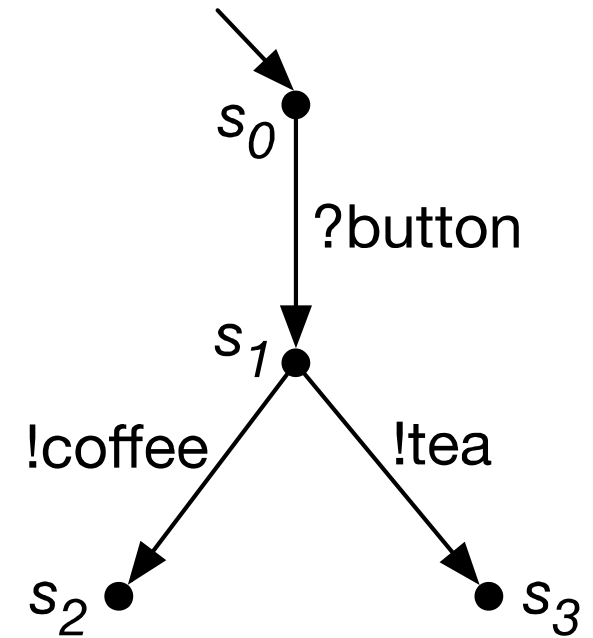


# choice

*declarations keep the same*

```
external 'extern'
process('tea-or-coffee') {
  # declarations of labels, variables
  timeout 10.0
  channel('extern') {
    stimulus 'button'
    responses 'tea', 'coffee'
  }

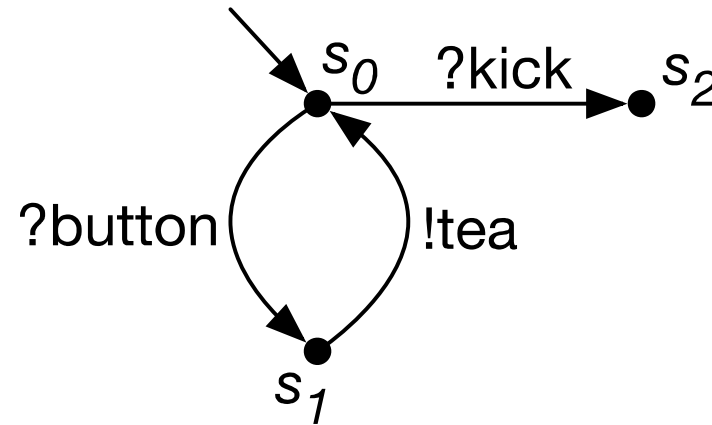
  # behaviour of the process
  receive 'button'
  choice {
    o { send 'tea' }
    o { send 'coffee' }
  }
}
```



# states and goto

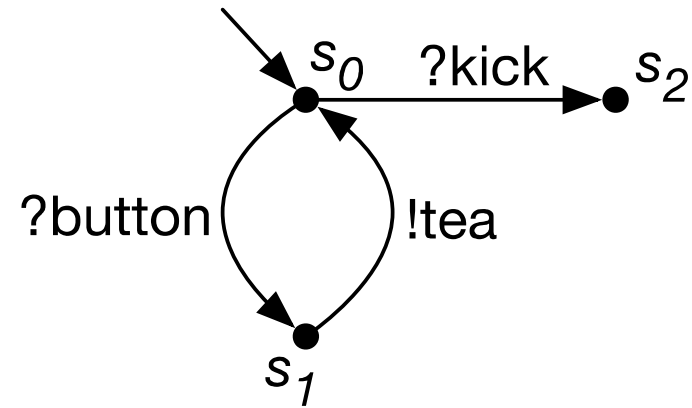
*convention:  
states start in  
column 1*

```
external 'extern'  
process('tea-or-kick-goto') {  
  timeout 10.0  
  channel('extern') {  
    stimuli 'button', 'kick'  
    response 'tea'  
  }  
  
  state 'loop'  
  choice {  
    o { receive 'kick' }  
    o { receive 'button'; send 'tea'; goto 'loop' }  
  }  
}
```



# repeat

```
external 'extern'  
process('tea-or-kick') {  
  timeout 10.0  
  channel('extern') {  
    stimuli 'button', 'kick'  
    response 'tea'  
  }  
  repeat {  
    o { receive 'kick' ; stop_repetition }  
    o { receive 'button'; send 'tea' }  
  }  
}
```



*break out of the loop*

# AML Quick Reference

For the 'Coffee Machine' exercise, only the following statements are needed:

- **send & receive** (without any options)
- **choice**
- **repeat**
- **state & goto**

```
external channel_name [, type: :duplex]
internal channel_name
...
process(process_name) {
  timeout <decimal_value>
  channel(channel_name) {
    stimulus label_name ,
      param_name => <type> ,
      ...
    response label_name, ...
  }
  var var_name, <type> [, <initial_value>]
  ...
  # behavior of process: AML statements
}

process(process_name) {
  ...
}
```

*channel\_name has to be declared globally*

*AML constructs are separated by newlines or ;*

`include 'model_part.aml'` *can be used anywhere*

```
<type>
:integer
:decimal
:boolean
:string
:date
:time
[ <type> ] list
{ <type> => <type> } hash
{ field_name => <type> , ... } struct
```

```
choice {
  o <statements>
  o <statements>
  ...
}
```

```
repeat {
  o <statements>
  o <statements>
  ...
}
```

```
deterministic choice
_if <bool_expr> ,
_then { ... } [,
_else { ... } ]
```

```
optionally { <statements> }
```

```
internal constraint
constraint <bool_expr>
```

```
internal action
update <assignment_expr>
```

```
send
receive
expedit
urge
aft
befo
constrai
upda
no
```

```
state state_name
goto state_name
stop_repetition
next_repetition
```

```
deterministic loop
_while(<bool_expr>) {
  ... # body
}
```

```
behavior(behavior_name, :terminating
, [param_name => <type>, ...] [, <return_type>] ) {
  # AML statements
}
terminating behaviors return value with exit_with
```

```
call terminating_name [, [<expr>, ...] [, into:<var_name>]
behave_as non_terminating_name [, [<expr>, ...]
```

```
Ruby code
def method(params)
  ...
end
if <ruby_expr>
  # AML fragments
else
  # AML fragments
end
"...#{<ruby_expr>}..."
```

- **Names** of labels, states, variables are 'strings' in quotes.
- Curly brackets { ... } can be used to group statements.
- Statements are separated by **newlines** (or ';').

```
=> <type>) { |p, ...|
...
}
AML expressions: constraints or updates
```

<b>fixed font</b>	keywords
*_name	string, e.g. "Notify"
<*_expr>	string, e.g. "x > 5"
<foo>	grammar placeholder
...	repetition / placeholder
[ ... ]	optionally

---

# Laboratory: Coffee Machine

- Modeling and testing a **beverage** machine, offering
  - **coffee, tea, and ... lemonade**
- First steps with the **Axini Modelling Platform** (AMP)

Goal is to make a **model of the SUT**.

- alternative, **high-level abstraction** of the SUT.
- **direction of messages** (stimuli, responses) is from the point of view of the SUT.



# Laboratory: Coffee Machine – HOWTO

## Model Based Testing with the Axini Modeling Platform – LABORATORY –

### Introduction

In this laboratory, you will carry out some practical exercises with a state-of-the-art Model Based Testing (MBT) tool: the Axini Modeling Platform (AMP). AMP is an industrial-strength tool developed by Axini<sup>1</sup> during the last decade.

The modeling language used in AMP is the Axini Modeling Language (AML). AML models are defined upon Symbolic Transition Systems (STS), a data-extension of Transition Systems (LTS). For this laboratory we will mostly use the LTS part of

AMP is an integrated development Platform for AML models. Models can be simulated, and most importantly, tested against a system under test (SUT). All test results are automatically saved. AMP uses an *online testing* approach, which means that test cases are generated on-the-fly while testing the SUT, rather than beforehand.

To become familiar with AMP, you will first do an introductory exercise with a Coffee Machine model and SUT. This will give you some hands-on experience with modeling and testing. Afterwards, you will be ready to do a small project with AMP, where you will model a remote controlled door (SMARTDOOR) from scratch using a specification document and test various implementations of that system.

Follow instructions:

1.1 Exploring AMP

1.2 Extending the model

1.3 Testing

1.4 Lemonade?!

*Ignore the references to the  
'SmartDoor' exercise.*

**axini**

---

Discussion and Evaluation of the  
**Coffee Machine**  
**Exercise**

---

# Coffee Machine

```
state 'start'
  choice {
    o { receive 'button_coffee' ; goto 'coffee' }
    o { receive 'button_tea' ; goto 'tea' }
    o { receive 'button_lemonade' ; goto 'lemonade' }
  }
```

```
state 'coffee'
  send 'coffee'
  goto 'start'
```

```
state 'tea'
  send 'tea'
  goto 'start'
```

```
state 'lemonade'
  choice {
    o { send 'lemonade' }
    o { send 'coffee' }
    o { send 'tea' }
  }
  goto 'start'
```

After *?button\_lemonade*, we observe either *!lemonade*, *!coffee*, or *!tea*.

Using *repeat* instead of states/goto.

```
repeat {
  o { receive 'button_coffee' ; send 'coffee' }
  o { receive 'button_tea' ; send 'tea' }
  o { receive 'button_lemonade'
    choice {
      o { send 'coffee' }
      o { send 'tea' }
      o { send 'lemonade' }
    }
  }
}
```

```
state 'start'
  choice {
    o { receive 'button_coffee' }
    o { receive 'button_tea' }
    o { receive 'button_lemonade' }
  }
  choice {
    o { send 'coffee' }
    o { send 'tea' }
    o { send 'lemonade' }
  }
  goto 'start'
```

Too loose: allowing too much behavior.

# Coffee Machine (exact?)

Using a **state variable**, which remembers the last beverage.

```
var 'last', :string, ''

repeat {
  o { receive 'button_coffee'; send 'coffee', update: "last = 'coffee'" }
  o { receive 'button_tea'; send 'tea', update: "last = 'tea'" }
  o {
    receive 'button_lemonade'
    choice {
      o { send 'lemonade', constraint: "last == ''" }
      o { send 'tea', constraint: "last == 'tea'" }
      o { send 'coffee', constraint: "last == 'coffee'" }
    }
  }
}
```

We also have to use the **update** and **constraint** options of a label here.

# Thank you!



### choice

```
external 'extern'
process('tea-or-coffee') {
  # declarations of labels, variables
  timeout 10.0
  channel('extern') {
    stimulus 'button'
    responses 'tea', 'coffee'
  }

  # behaviour of the process
  receive 'button'
  choice {
    o { send 'tea' }
    o { send 'coffee' }
  }
}
```

declarations keep the

### Coffee Machine

```
state 'start'
choice {
  o { receive 'button_coffee'; goto 'coffee' }
  o { receive 'button_tea'; goto 'tea' }
  o { receive 'button_lemonade'; goto 'lemonade' }
}

state 'coffee'
send 'coffee'
goto 'start'

state 'tea'
send 'tea'
goto 'start'

state 'lemonade'
choice {
  o { send 'lemonade' }
  o { send 'coffee' }
  o { send 'tea' }
}
goto 'start'
```

Using repeat instead of states/goto.

```
repeat {
  o { receive 'button_coffee'; send 'coffee' }
  o { receive 'button_tea'; send 'tea' }
  o { receive 'button_lemonade'
    choice {
      o { send 'coffee' }
      o { send 'tea' }
      o { send 'lemonade' }
    }
  }
}
```

After ?button\_lemonade, we observe either !lemonade, !coffee, or !tea.

```
state 'start'
choice {
  o { receive 'button_coffee' }
  o { receive 'button_tea' }
  o { receive 'button_lemonade' }
}

choice {
  o { send 'coffee' }
  o { send 'tea' }
  o { send 'lemonade' }
}
goto 'start'
```

Too loose: allowing too much behavior.

---

# Clients come to us for



- Highest quality possible



- Lower time to production (30% and more)



Ctrl

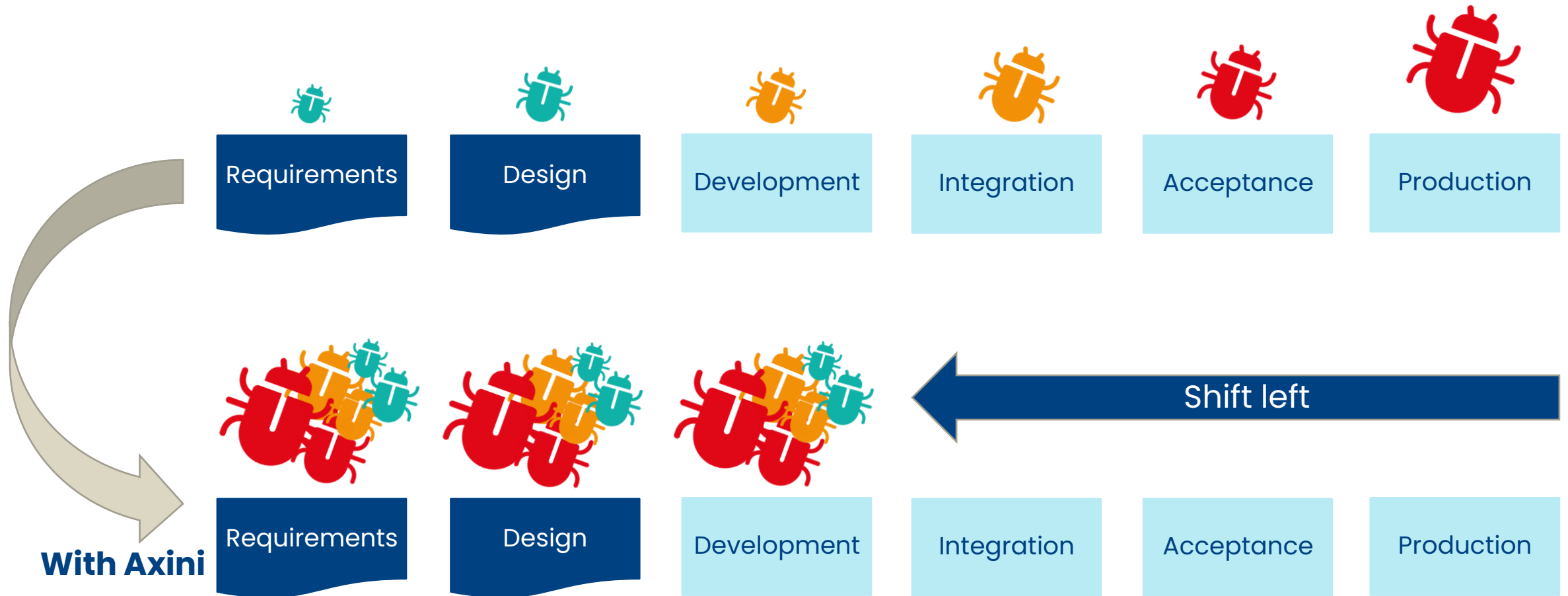
- Project control. No errors late in the process.  
Deliver what you promise.



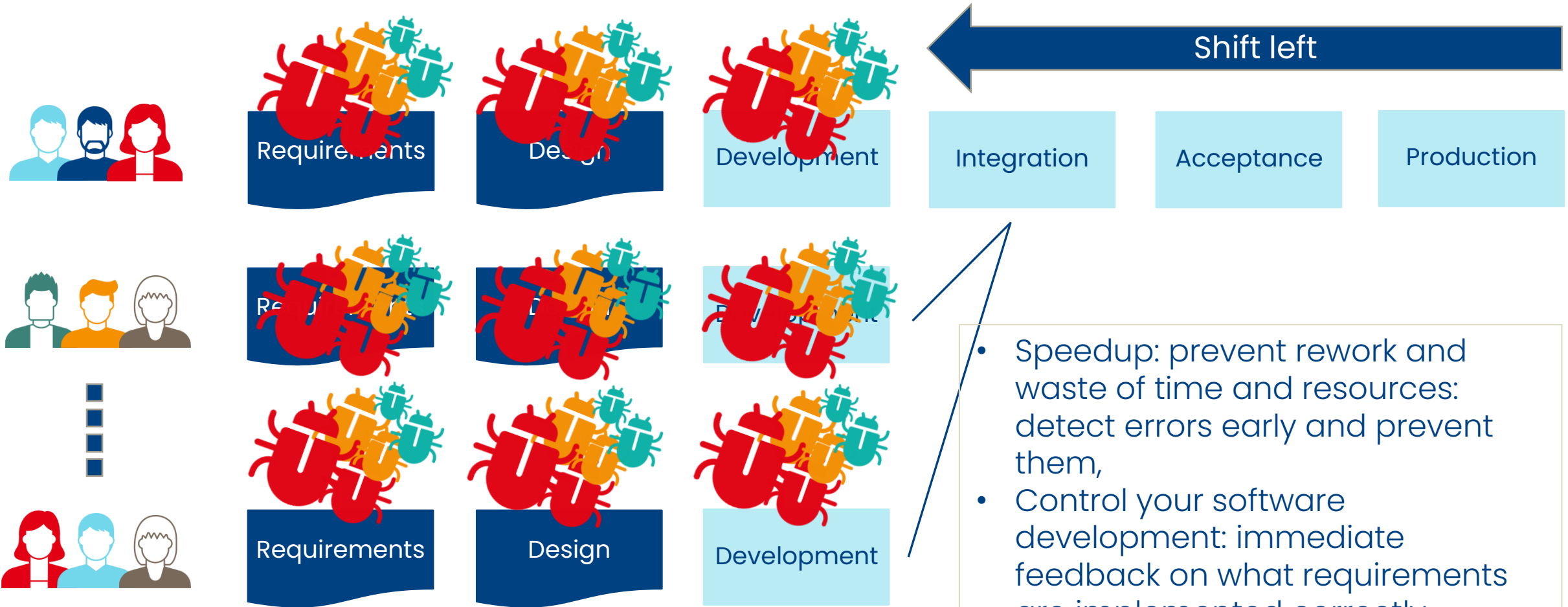
- Communication between business and IT



# How do we help?



# Axini platform and approach scales





# axini

---

## How to shift left?

---

# Axini

- Our **dream** is to optimize the **entire** software development process.
- Our **current** offering **optimizes** the verification and validation (V&V) of systems. From a language to write down requirements all the way to automated testing.
- We offer a platform that automates V&V: test-automation without the need to program test-scripts and test-data.
- We are a technology partner (no/limited consultancy). We work together with consultancy partners or directly with clients.
- We are primarily active in Finance, Rail and High-tech.

# Behavior-Driven Development (BDD)

- BDD is a Test-First, Agile practice that defines and automates tests as part of **specifying system behavior**.
- BDD is a collaborative process that creates a shared understanding of requirements between the **business and the Agile Teams**.
- BDD tests are business-facing scenarios that attempt to describe the behavior of a Story, Feature, or Capability from a **user's perspective**.
- These tests ensure that the system **continuously meets the specified behavior** even as the system evolves.

---

# Example Cucumber

## Feature: Managing users

As an admin

I am able to add new users

### Background:

Given I am logged in as an administrator

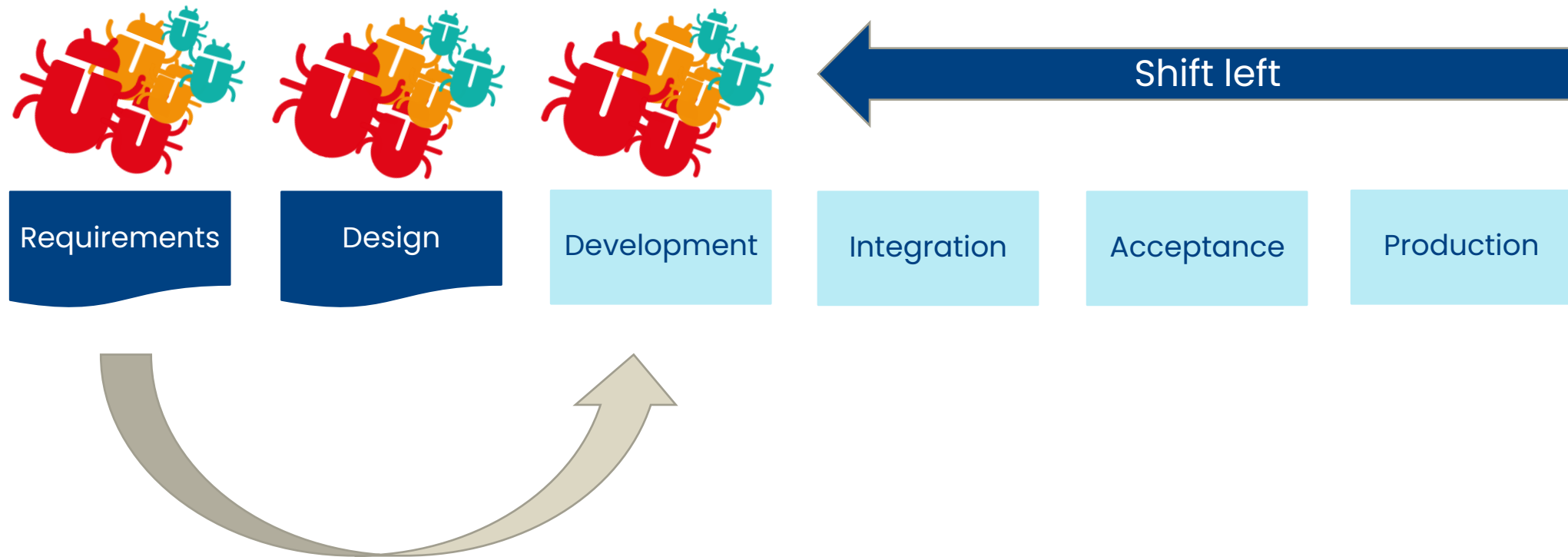
And I go to the users page

---

**axini** Scenario: Adding a new user

When I choose to add a new user

# BDD: Couples requirements and test



**axini**

---

# What is Axini Model Based Testing?