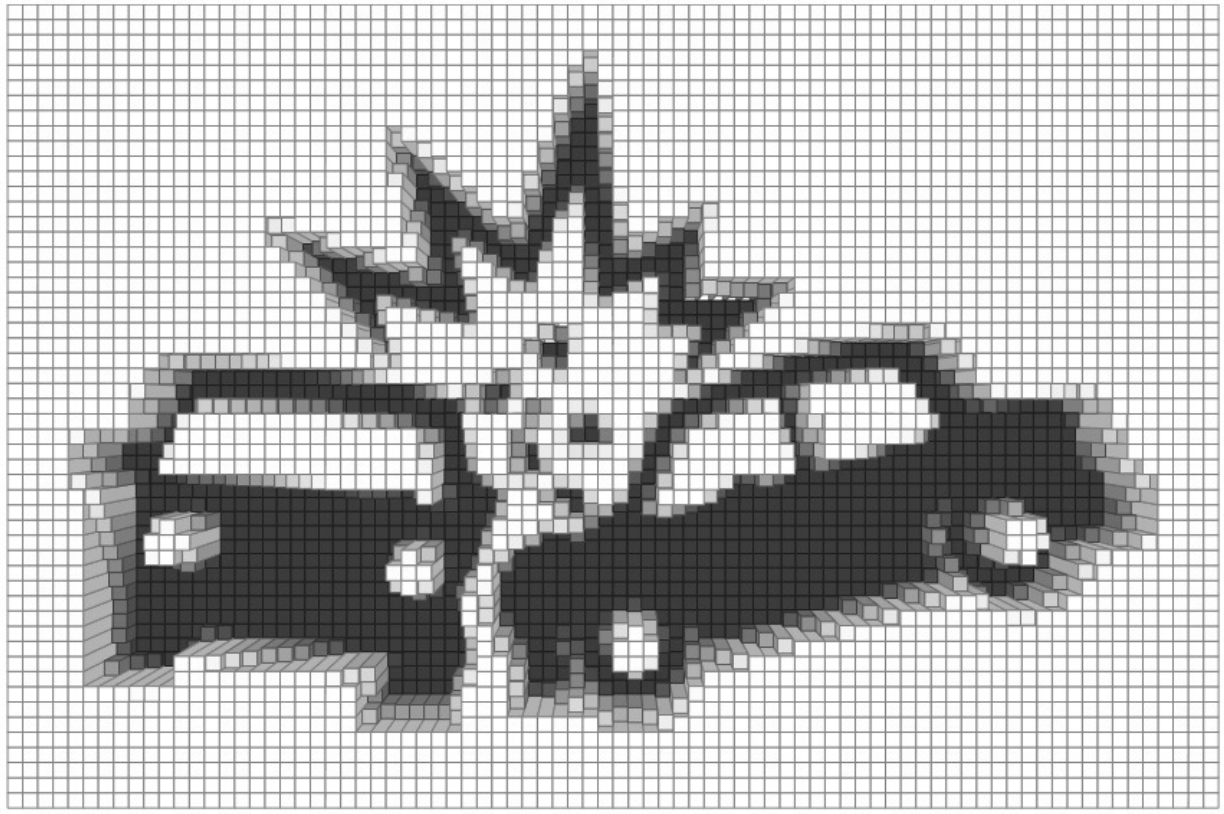
The future of models in testing

Safely crash in virtual space

SIOUX TECHNOLOGIES

Testdag



Bryan Bakker

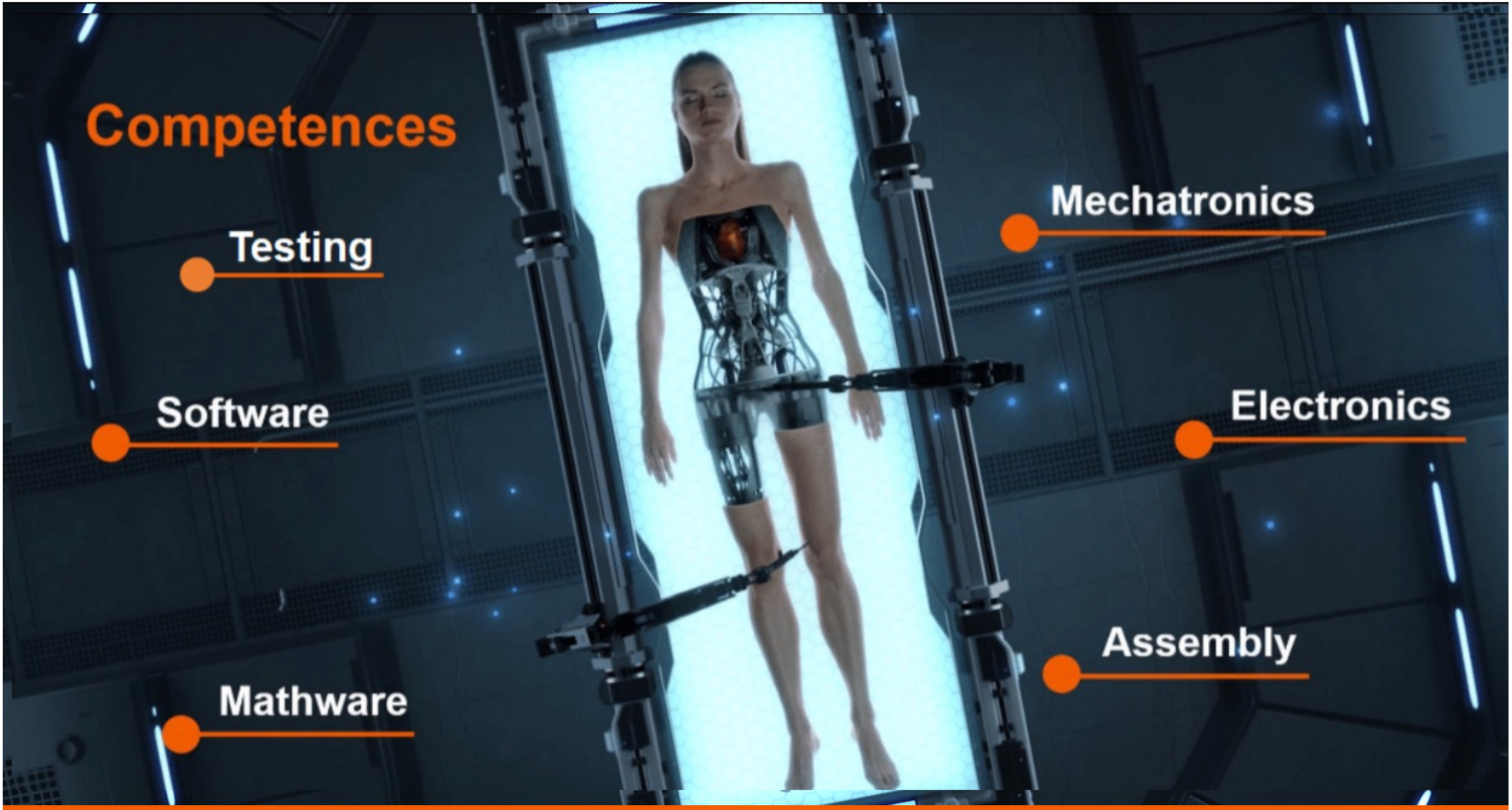
Dirk Coppelmans

Bryan Bakker

- Test Architect
- Tutor of several test related courses
- Domains: medical systems, professional security systems, semicon-industry, electron microscopy, material handling
- Specialties: test automation, integration testing, design for testability, reliability testing

Dirk Coppelmans

- Test Architect
- Test strategy, infrastructure & automation
- Consumer products, medical devices & industrial machinery
- Quote: “There is always one more bug”


Competences
Testing
Software
Mathware
Mechatronics
Electronics
Assembly

The facts

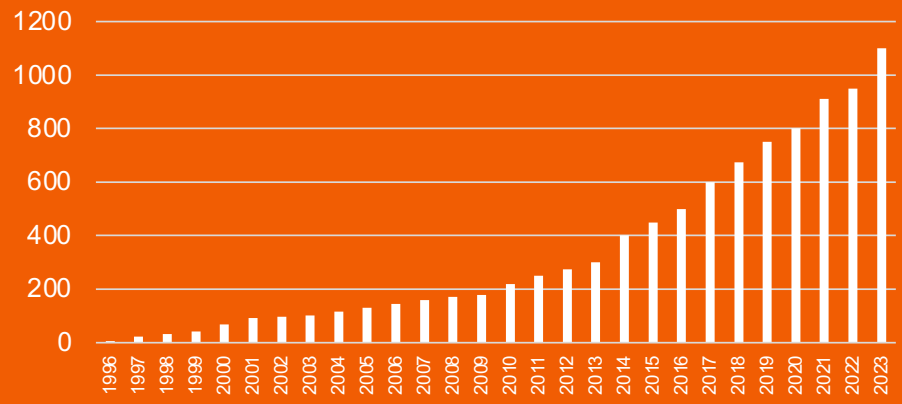
Annual turnover
€ 100,000,000

Customer satisfaction
8

Employees
1100+

Employee satisfaction
8

Number of employees



The future of models in testing

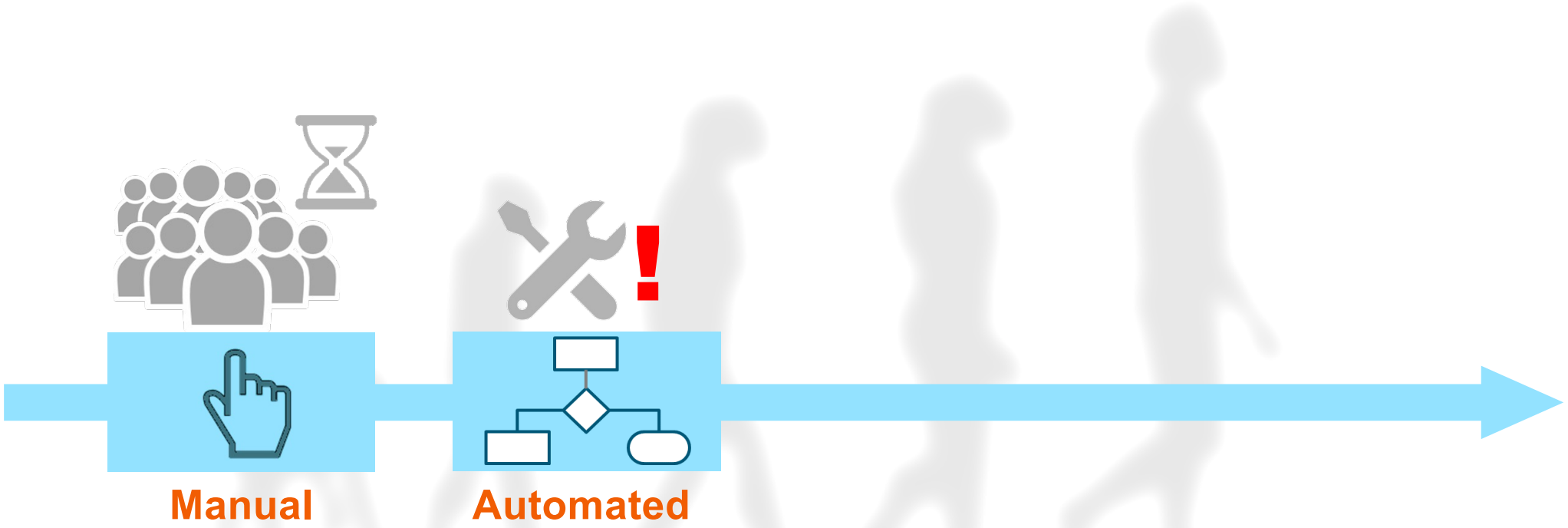
1. Evolution of software testing
2. Models in testing
3. Promises for the future

Evolution of software testing

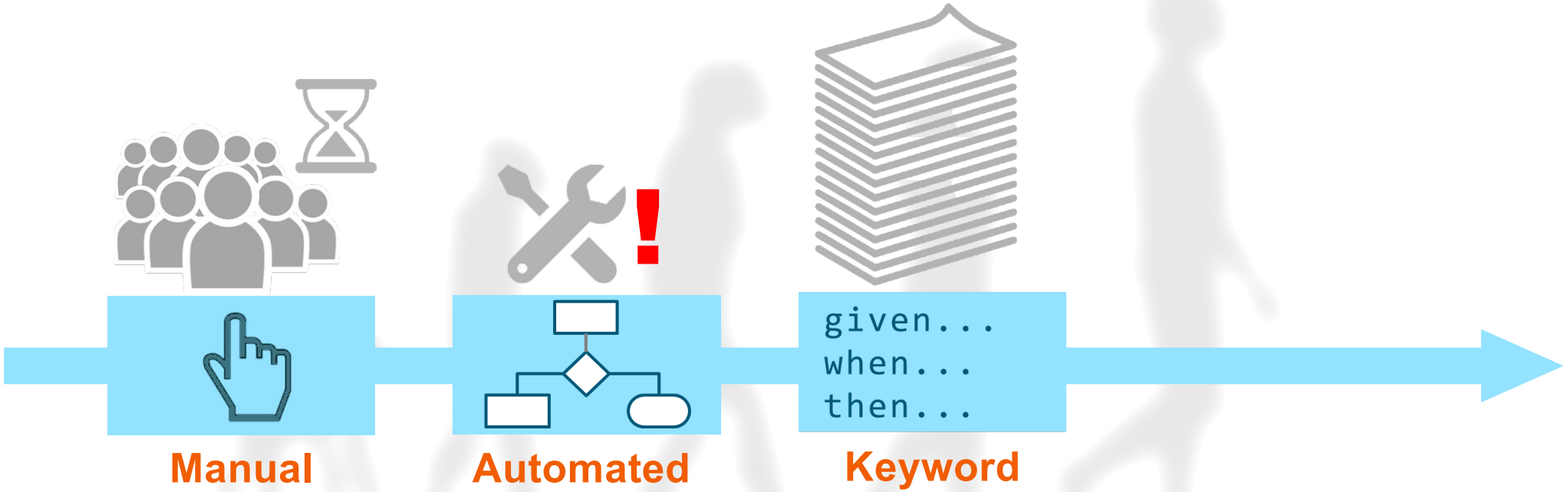
Evolution of software testing

Evolution of software testing

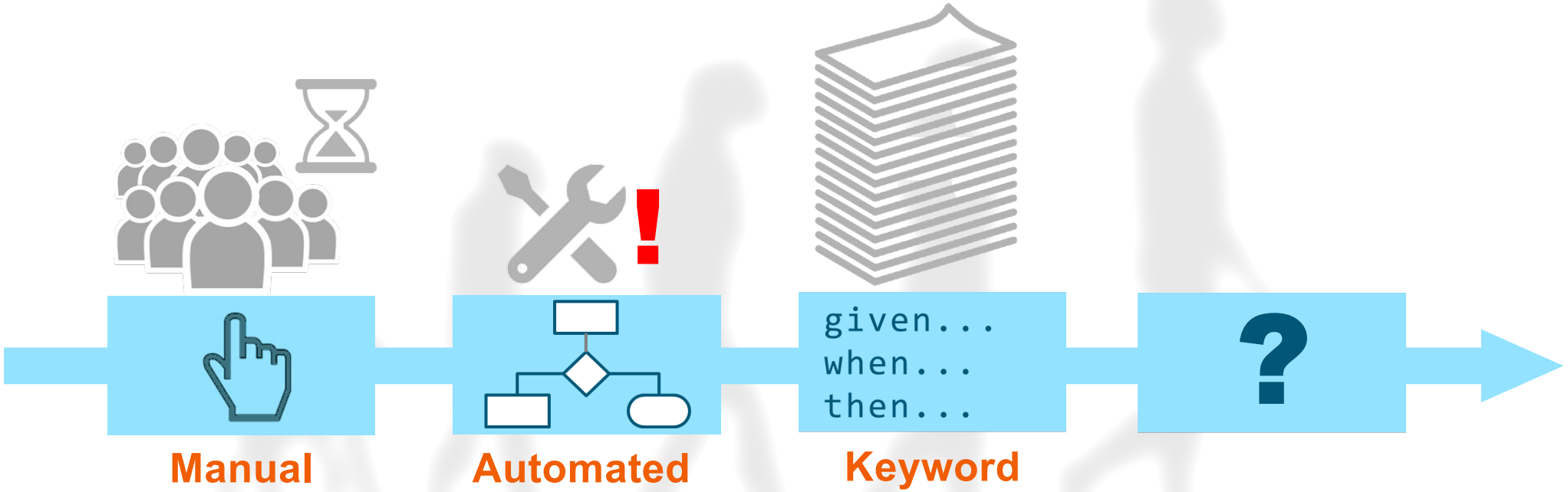
Evolution of software testing



Evolution of software testing



Evolution of software testing

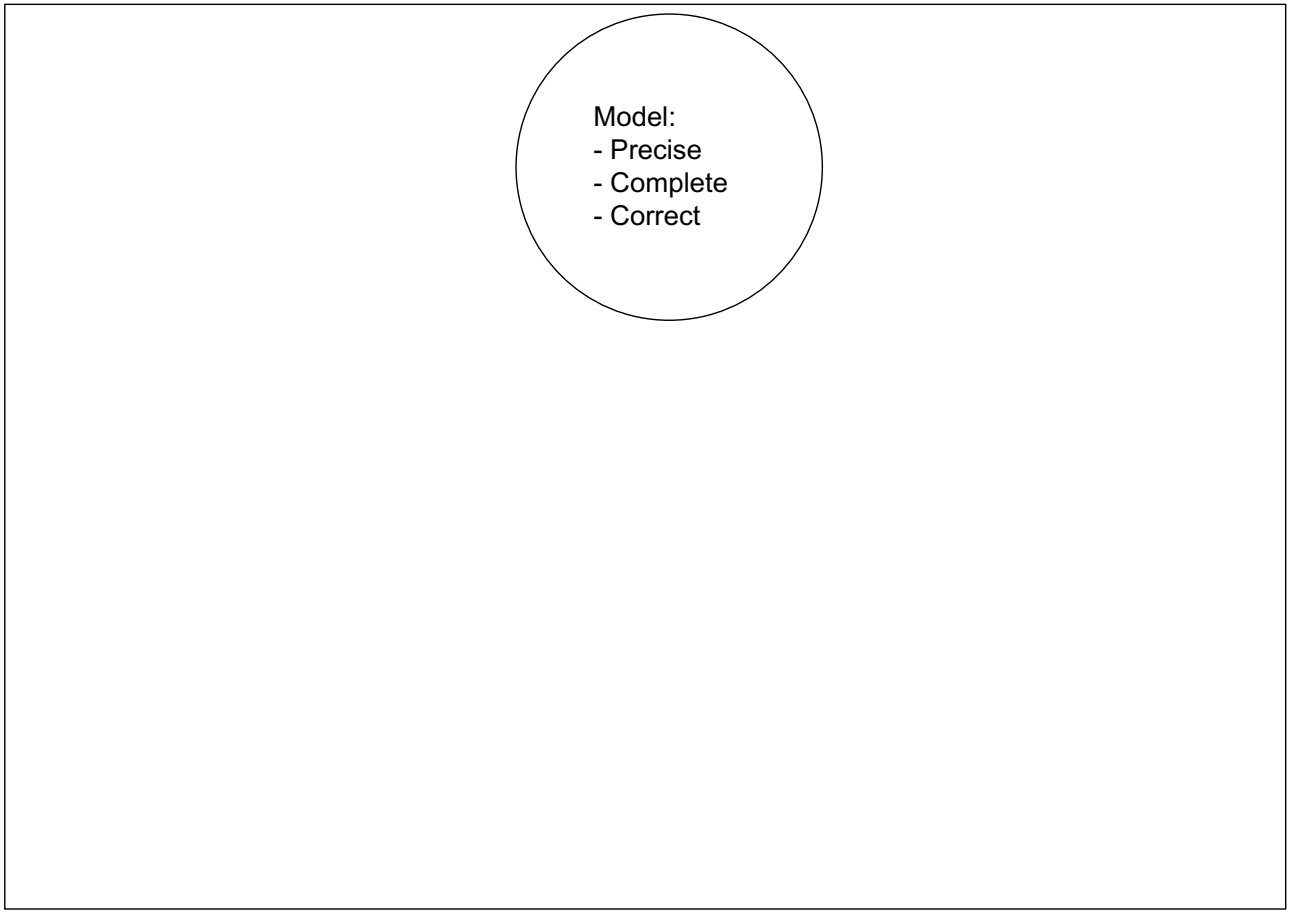


Evolution of software testing – Compare to SW development

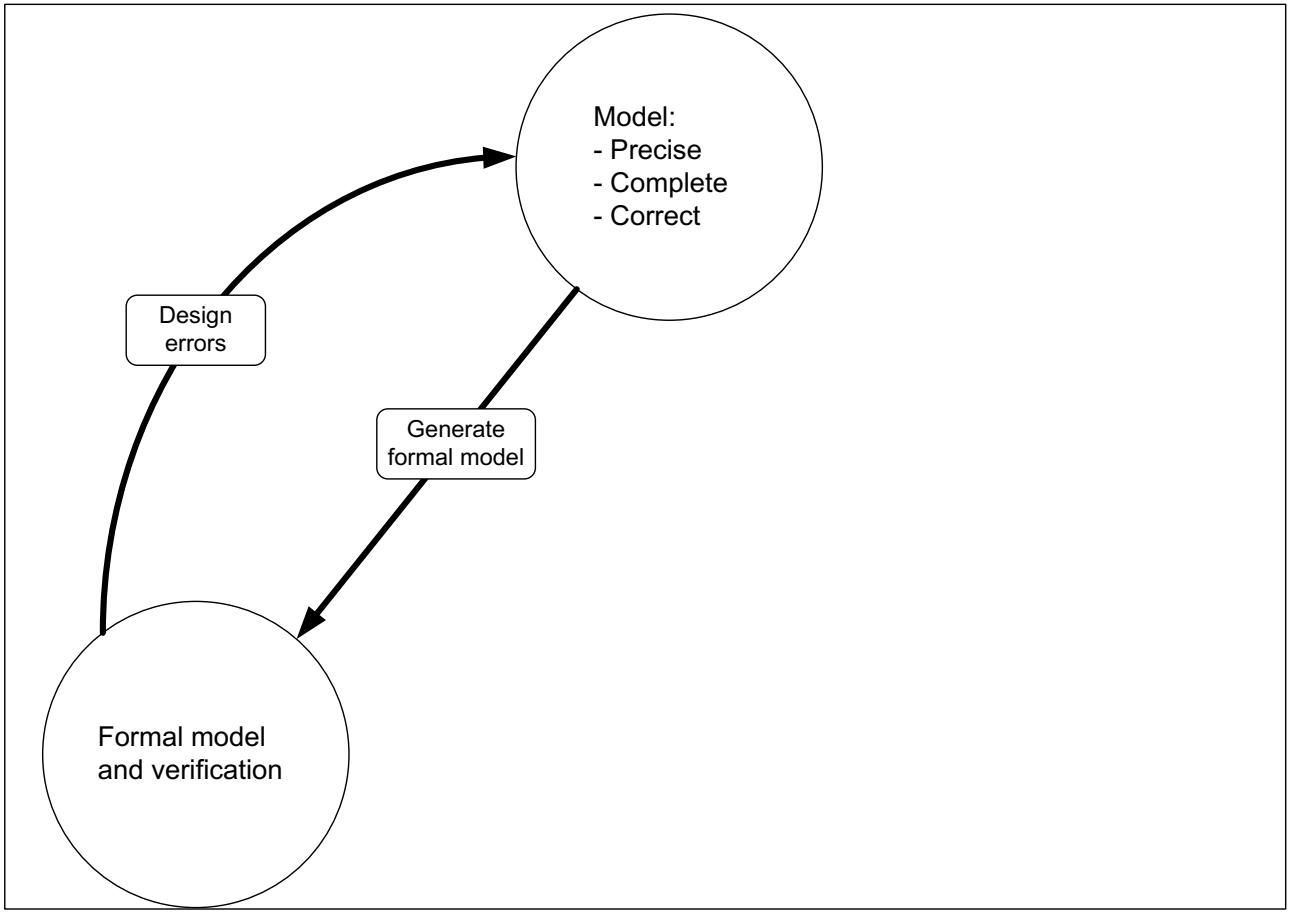
- Increase in use of formal models
- Originates from research & universities
- No longer limited to safety and reliability critical environments, like automotive and aviation
- Applied to manage complexity

Type	Tools
System design	• ASD / Dezyne (Verum) • mCRL2 (verification engine for ASD/Dezyne) • Cocotec
Interface	• Pact • ComMA

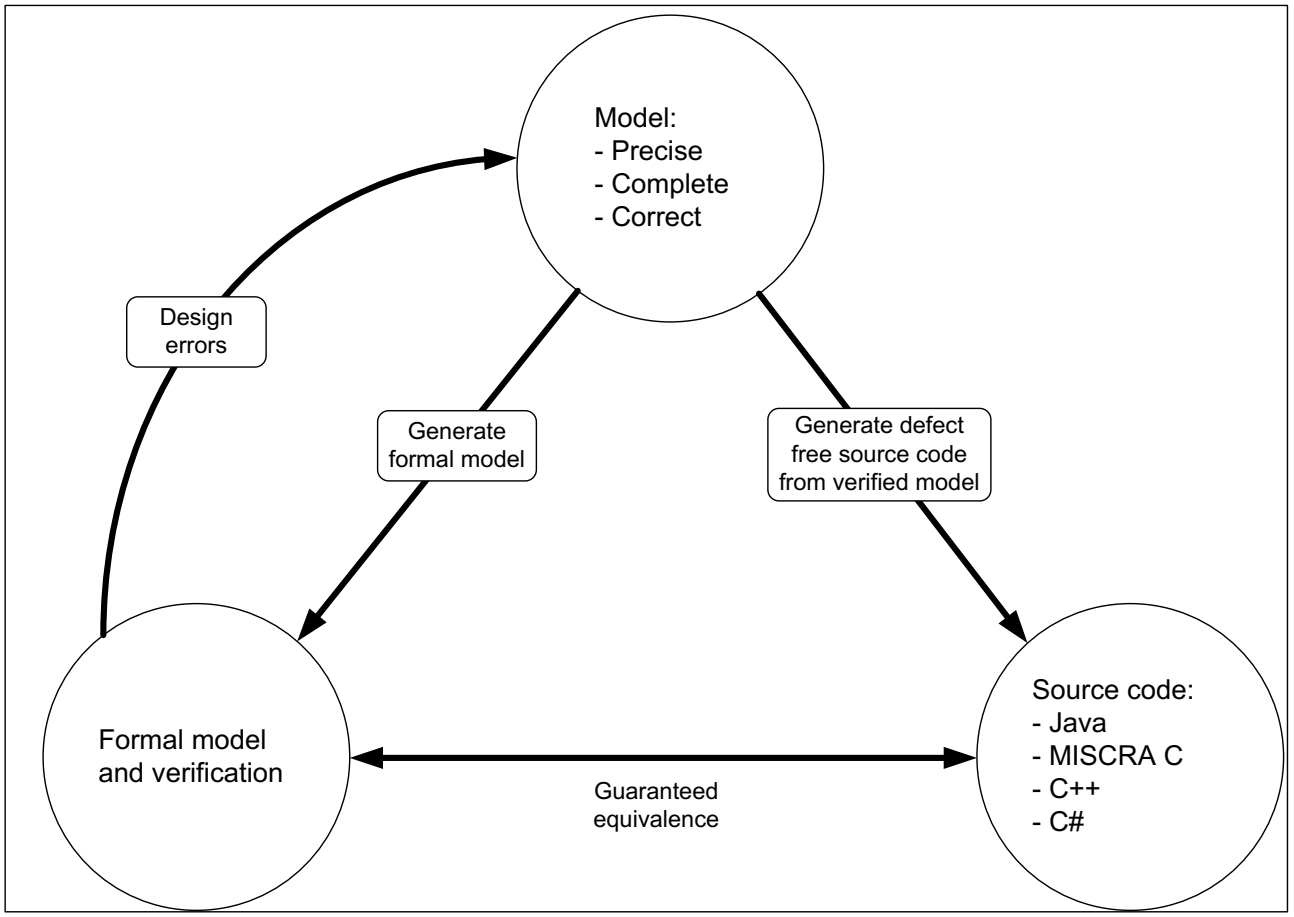
Model Driven Development (MDD) - Model



MDD – Model verification



MDD – Code generation



MDD – Experiences

- Quality of generated code very high
  - Especially reliability and stability
  - Functionality can still be wrong (also wrong in model)
  - No more programming errors like deadlocks, livelocks, starvation, race-conditions
- Integration with other parts still important

References:

- R. van Beusekom, J.F. Groote, P. Hoogendijk, R. Howe, W. Wesselink, R. Wieringa, T.A.C. Willemse. *Formalising the Dezyne Modelling Language in mCRL2*
- www.verum.com

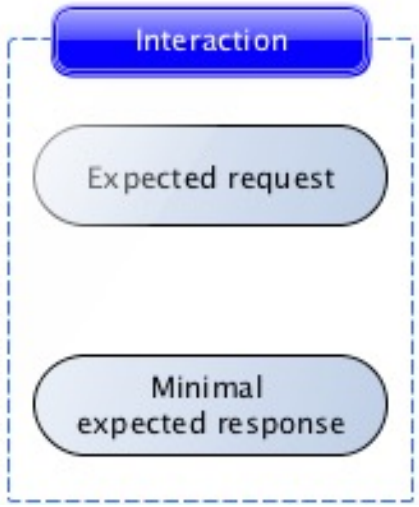
Models in testing

Interface modeling – Contract testing

- Consumer driven contracts
- In micro-service architecture
- Useful for interface definition
- True power: interface evolution
- Tool support e.g.: Pact Broker, Pactflow, Spring Cloud Contract, Schemathesis

References:
- https://pact.io/
- https://pactflow.io/
- https://spring.io/projects/spring-cloud-contract
- https://github.com/schemathesis/schemathesis

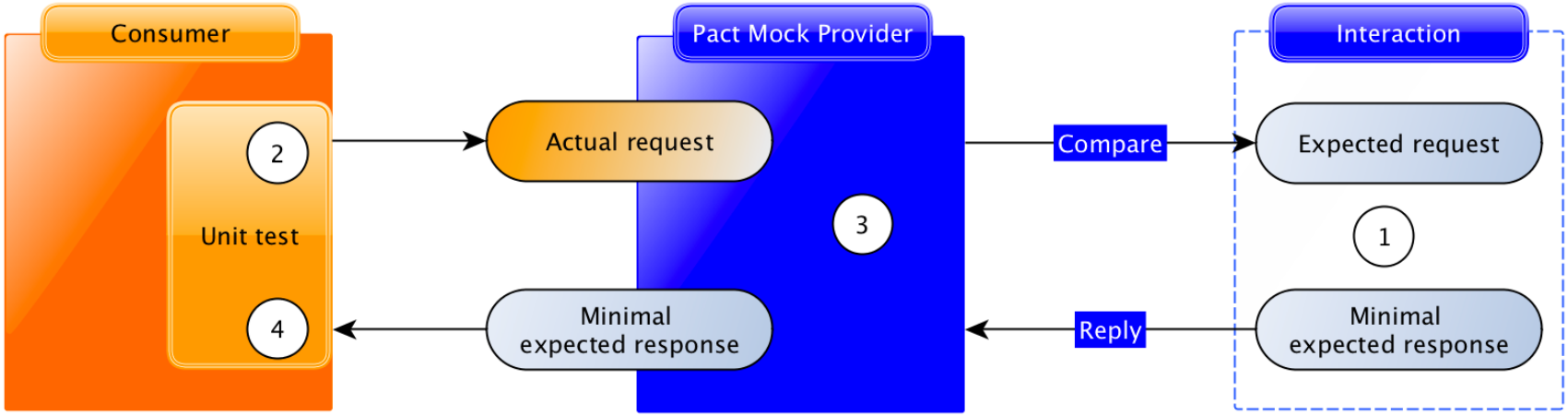
Contract



- **Consumer**: client that requires data
- **Provider**: service that provides data
- **Contract** (aka pact): collection of interactions:
  - **expected request:** what consumer needs to send to provider
  - **minimal expected response:** elements provider needs to return

https://pact.io/

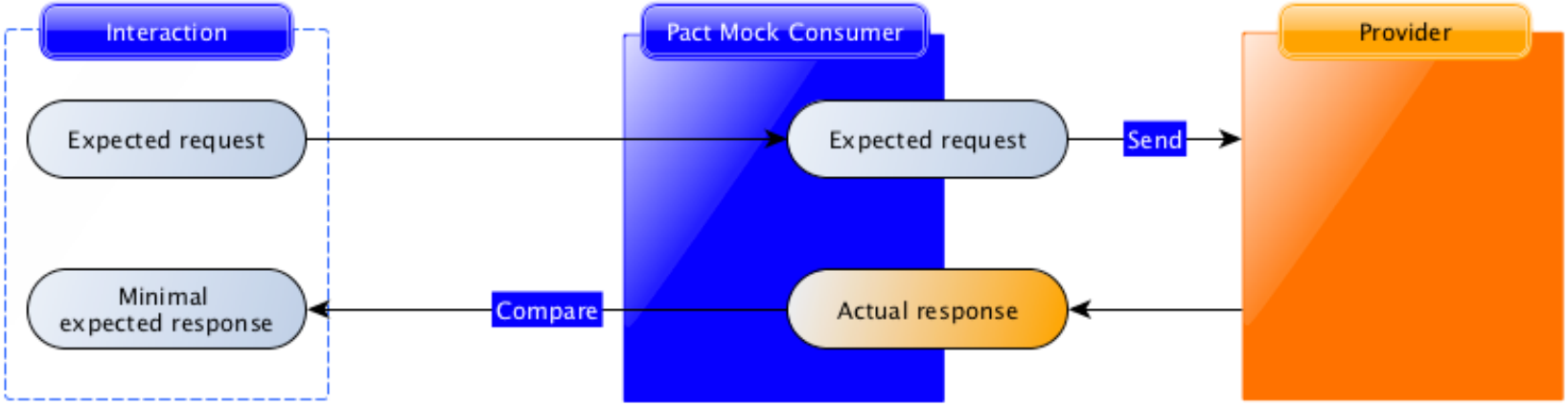
Consumer side testing



1. Using the Pact DSL, the expected request and response are registered with the mock service.
2. The consumer test code fires a real request to a mock provider (created by the Pact framework).
3. The mock provider compares the actual request with the expected request, and emits the expected response if the comparison is successful.
4. The consumer test code confirms that the response was correctly understood

https://pact.io/

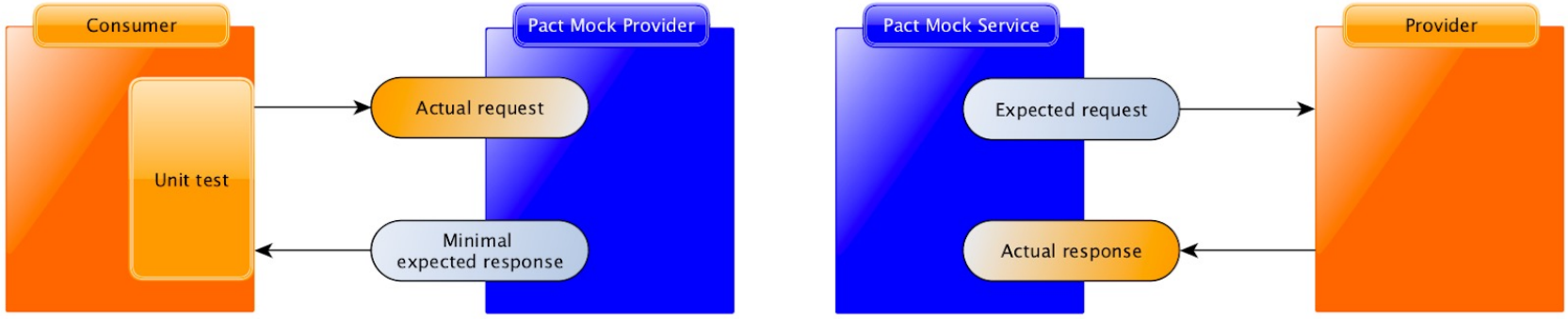
Provider side testing



- In provider verification, each request is sent to the provider, and the actual response it generates is compared with the minimal expected response described in the consumer test.
- Provider verification passes if each request generates a response that contains at least the data described in the minimal expected response.

https://pact.io/

Combination

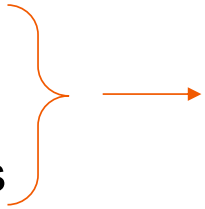


If we pair the test and verification process for each interaction, the contract between the consumer and provider is fully tested without having to spin up the services together.

https://pact.io/

ComMA
Component Modeling and Analysis

- Advanced interface modeling
- Developed by Philips Healthcare + TNO-ESI (now open source)
- Model consists of:
  - Signature
  - Behavior
  - Time & Data constraints



- Generated:
  - Visualization
  - Documentation
  - Interface code
  - Runtime Monitoring and interface conformance
  - Test cases

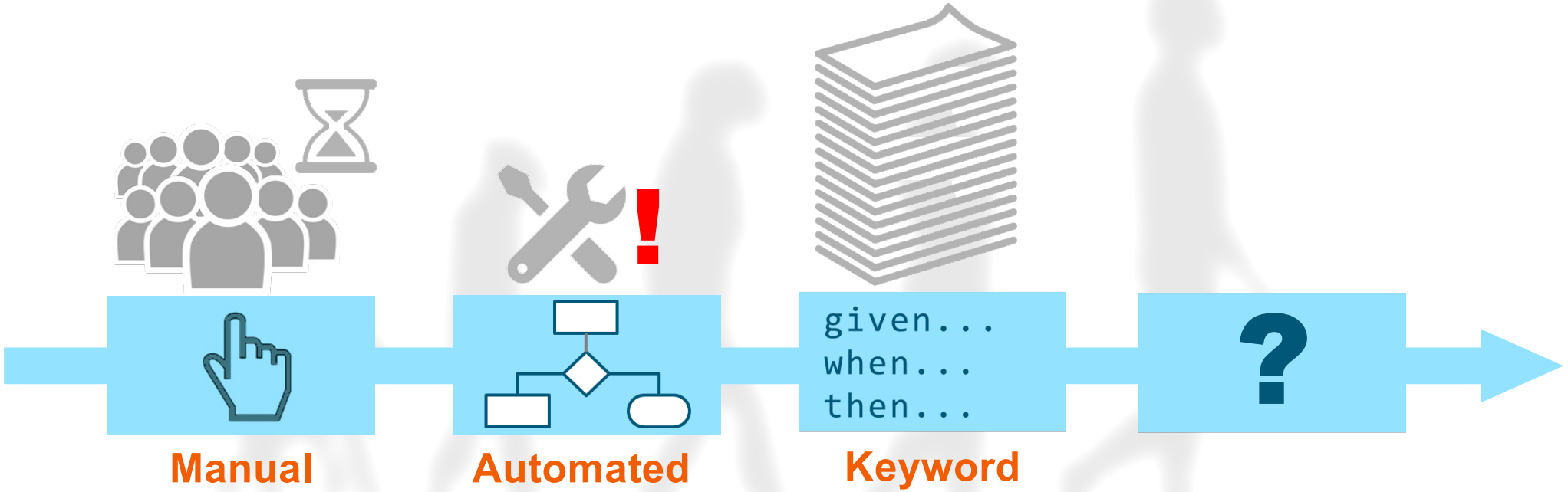
References:
- https://projects.eclipse.org/projects/technology.comma
- https://esi.nl/research/output/tools/comma

Design & Interface models

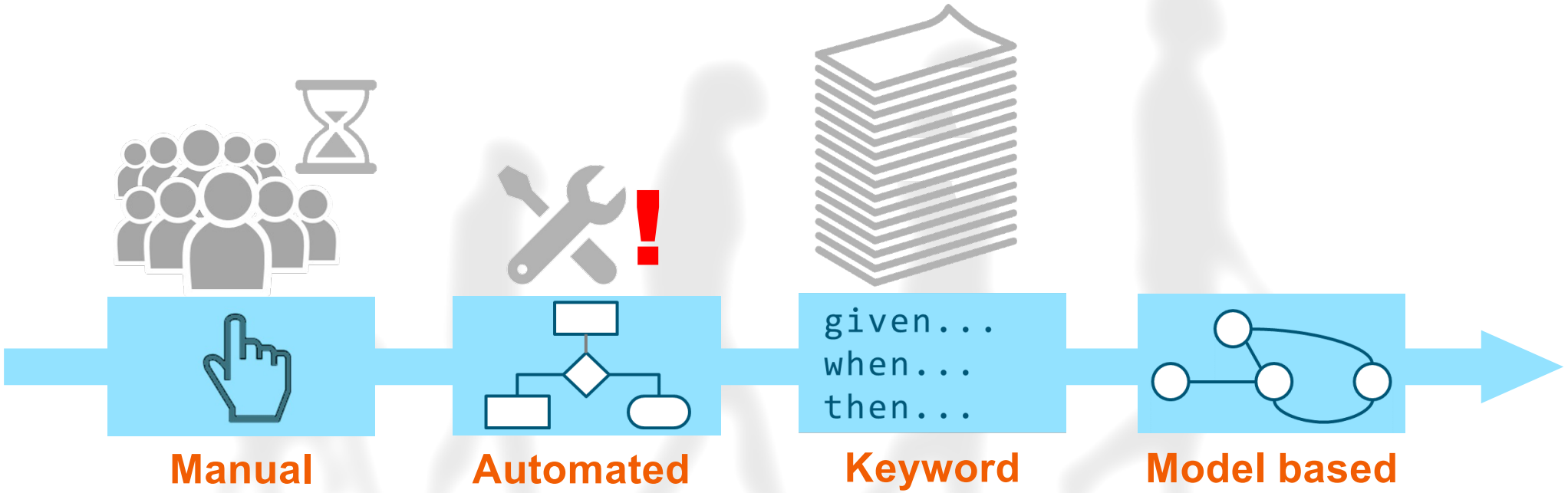
- MDD Design models are by far flawless
  - Often only complex+critical parts of the system modeled
- Interface models are
  - Rigorous
  - Valuable for interface evolution
  - Limited to interfaces

- By modeling behavior, new test possibilities arise

Evolution of software testing



Evolution of software testing



MBT Definition *)

Model based testing (MBT) is

automated test generation and execution

based on an abstract

behavior model

*) As used by Sioux

Developing a behavior model

1. Derive the behavior model from the requirements
2. Construct a behavior model based on collected field data (process mining)

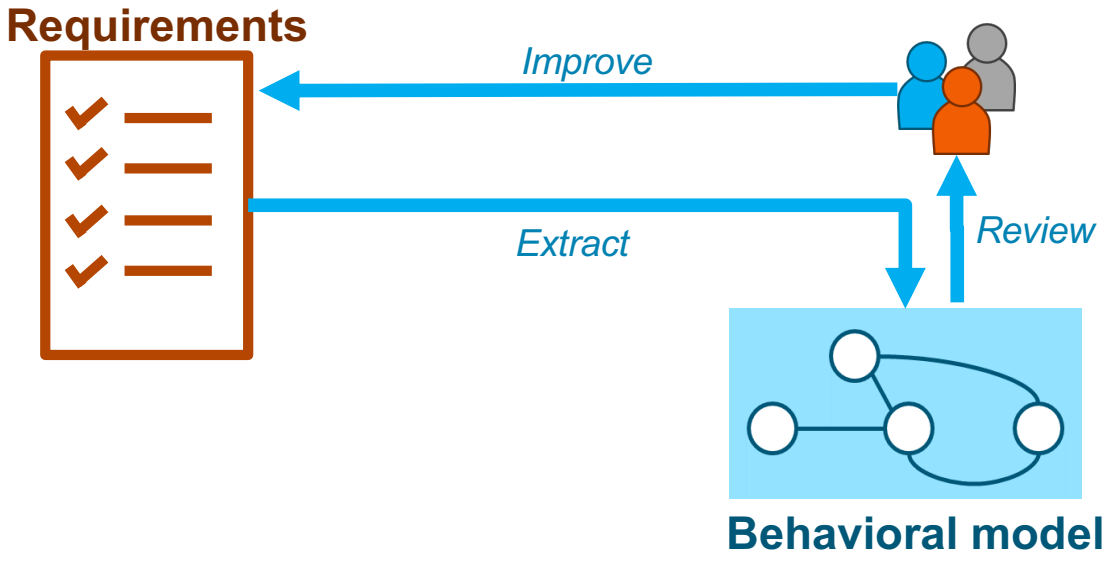
Derive from the requirements

Requirements

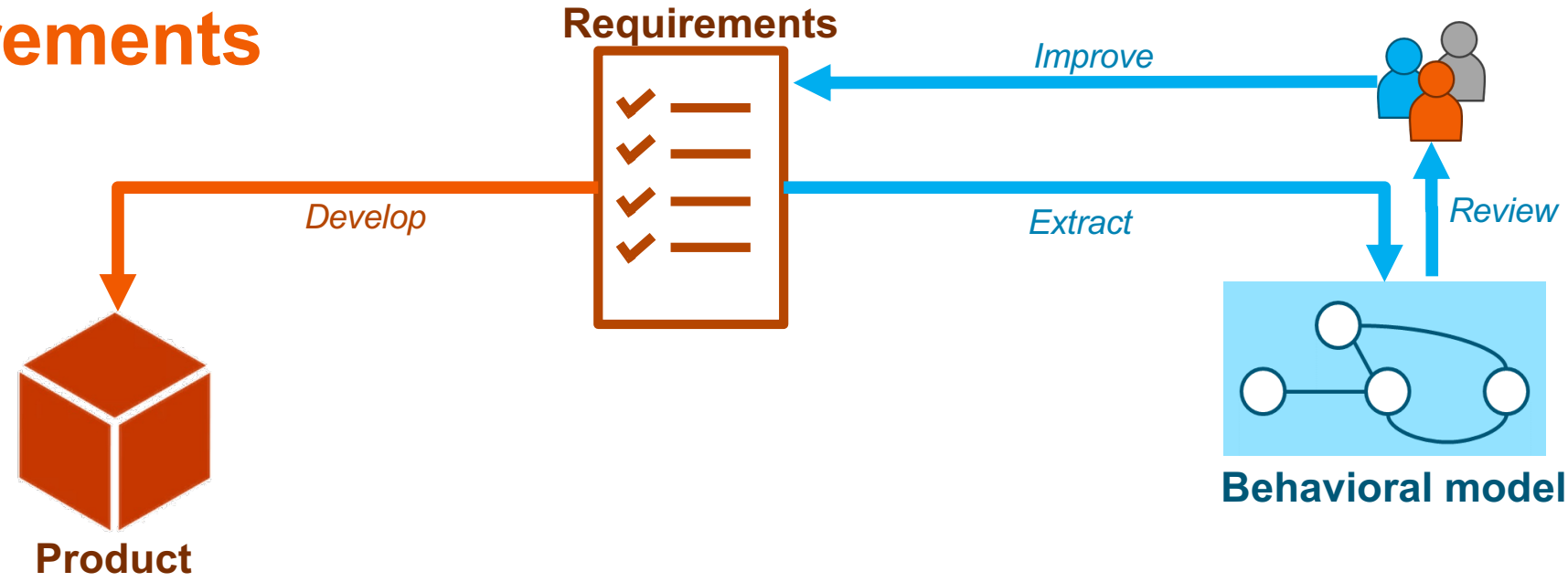
Development

Test

Derive from the requirements

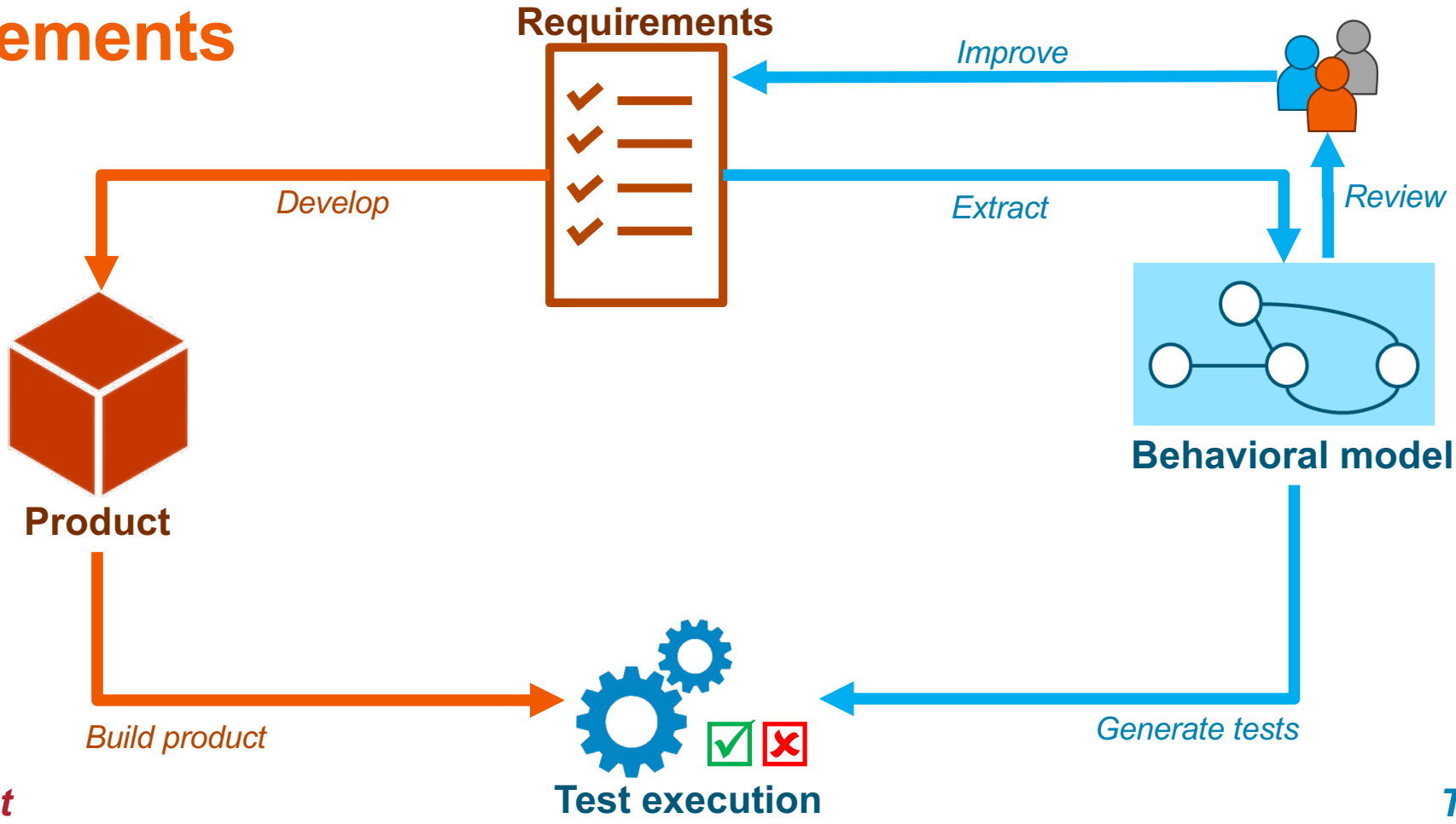


Derive from the requirements

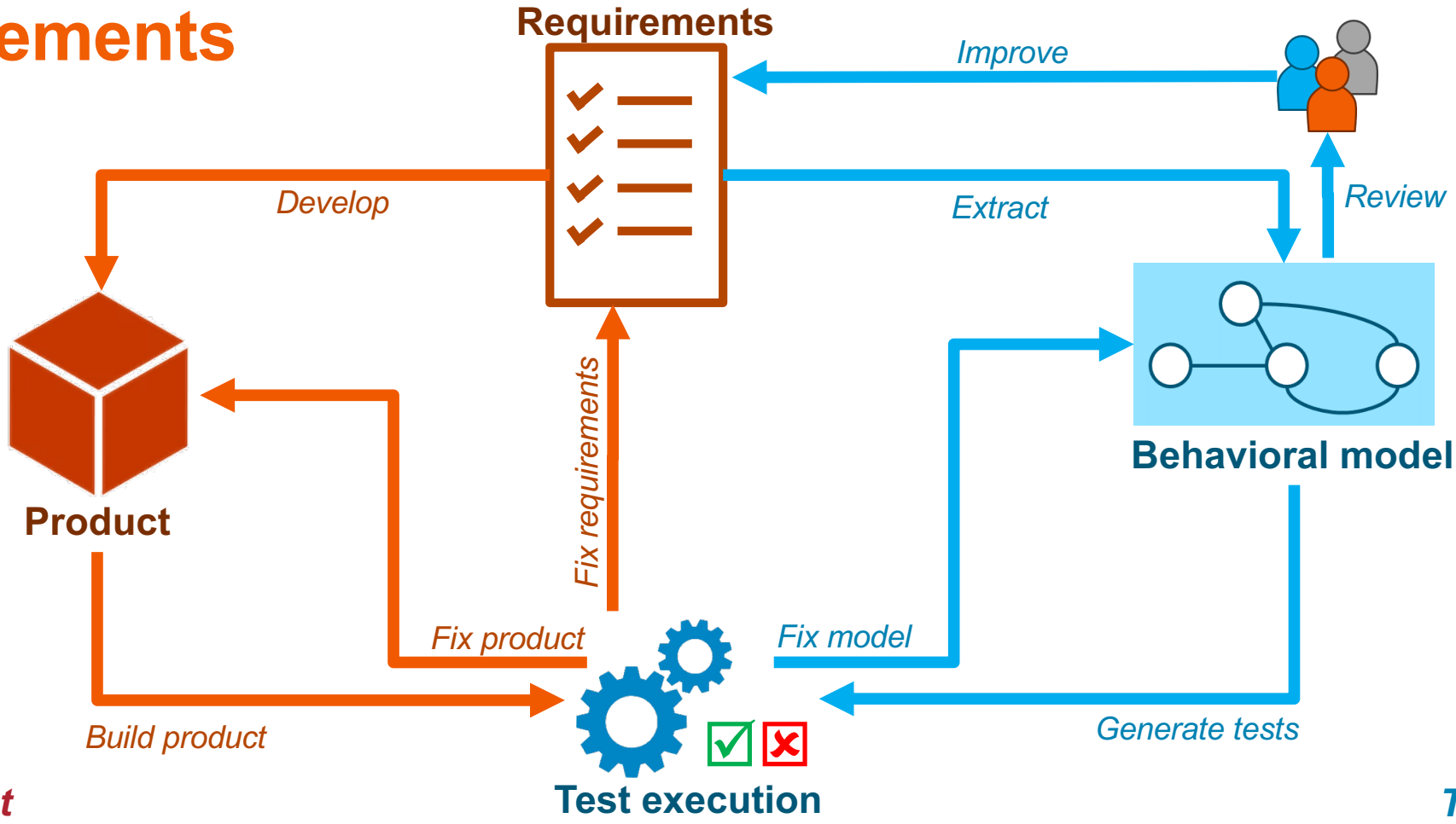


Development *Test*

Derive from the requirements



Derive from the requirements

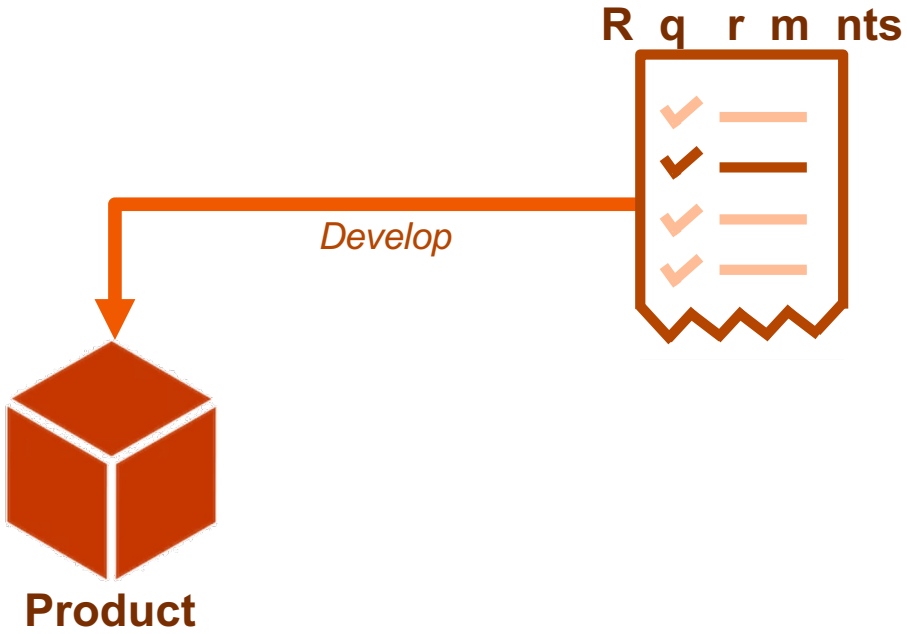


The next best thing – Developing a behavior model

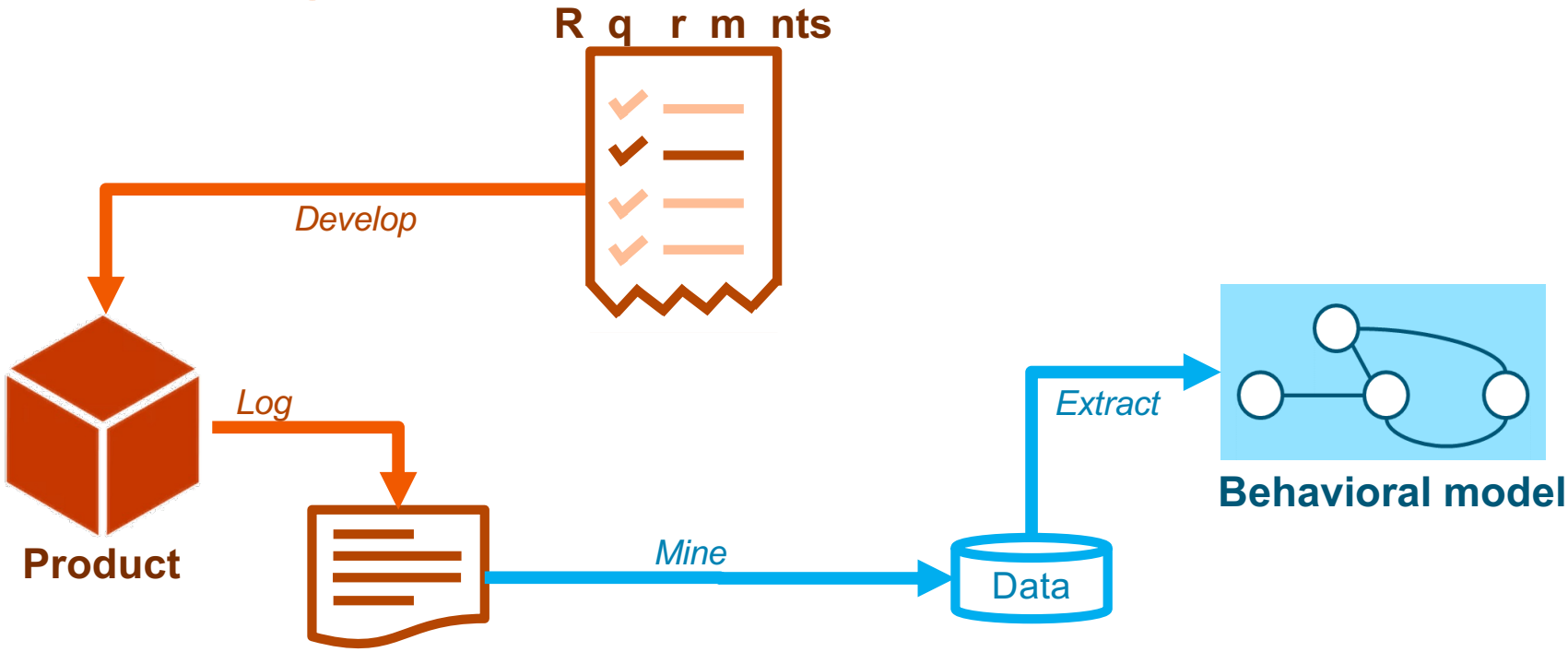
1. Derive the behavior model from the requirements
2. **Construct a behavior model based on collected field data (process mining)**

Process mining

Process mining

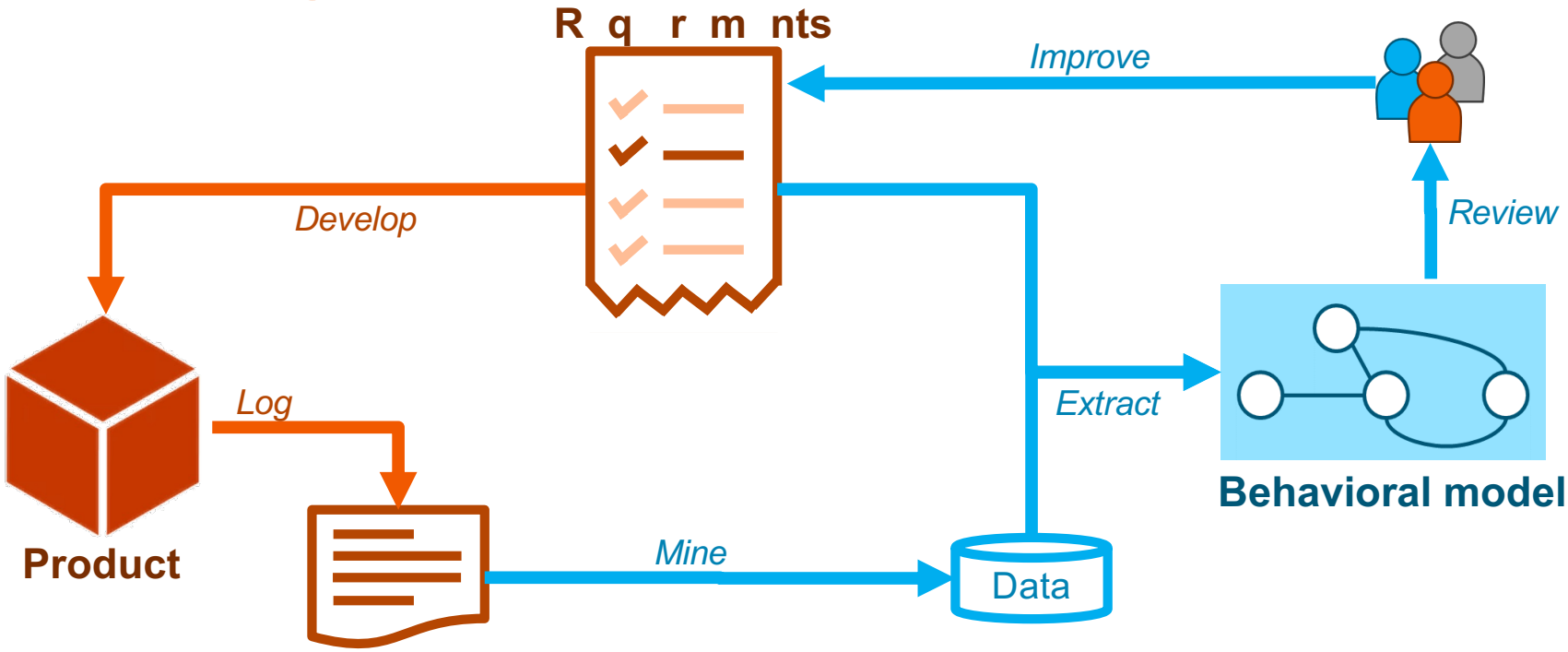


Process mining



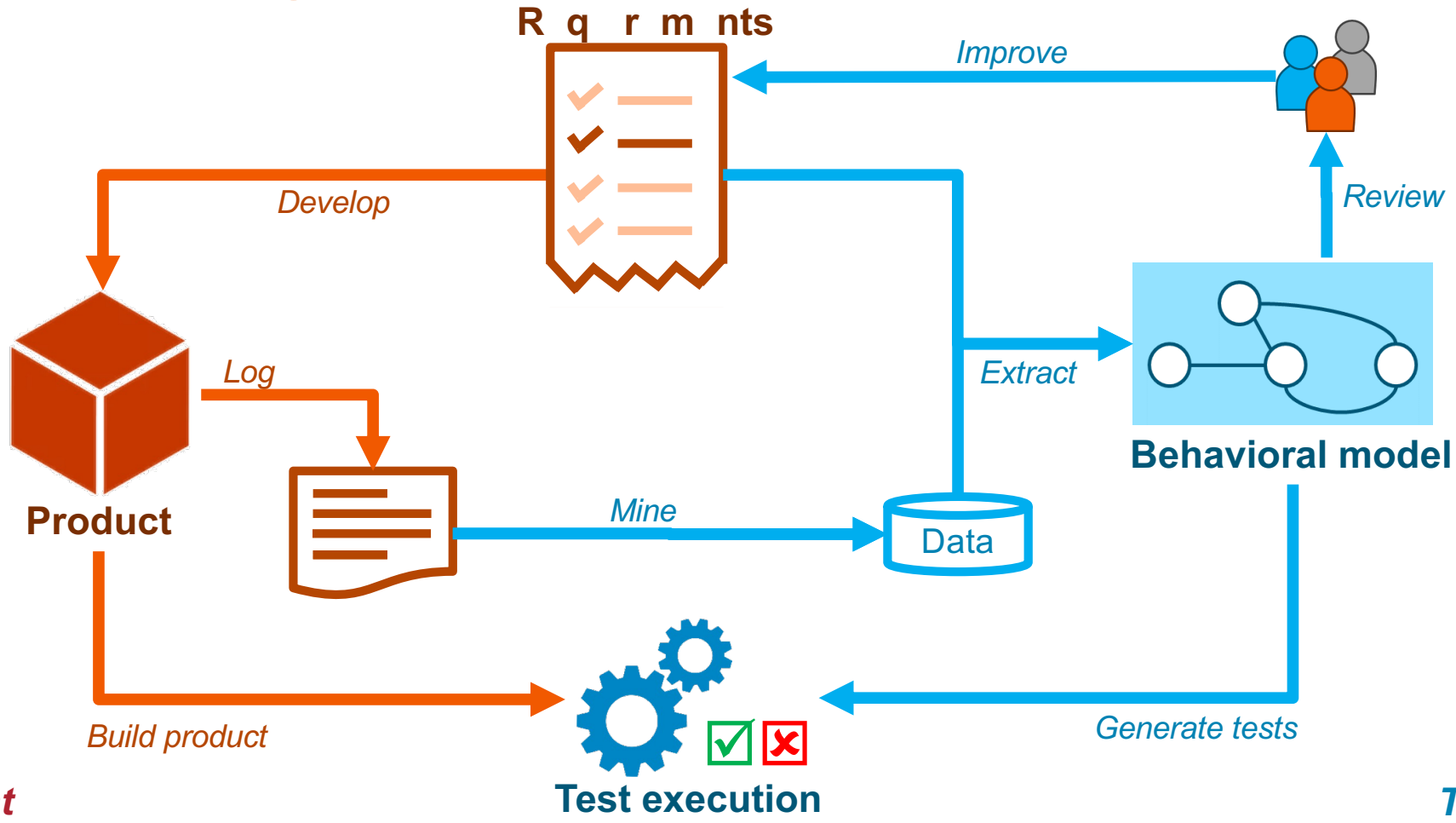
Development *Test*

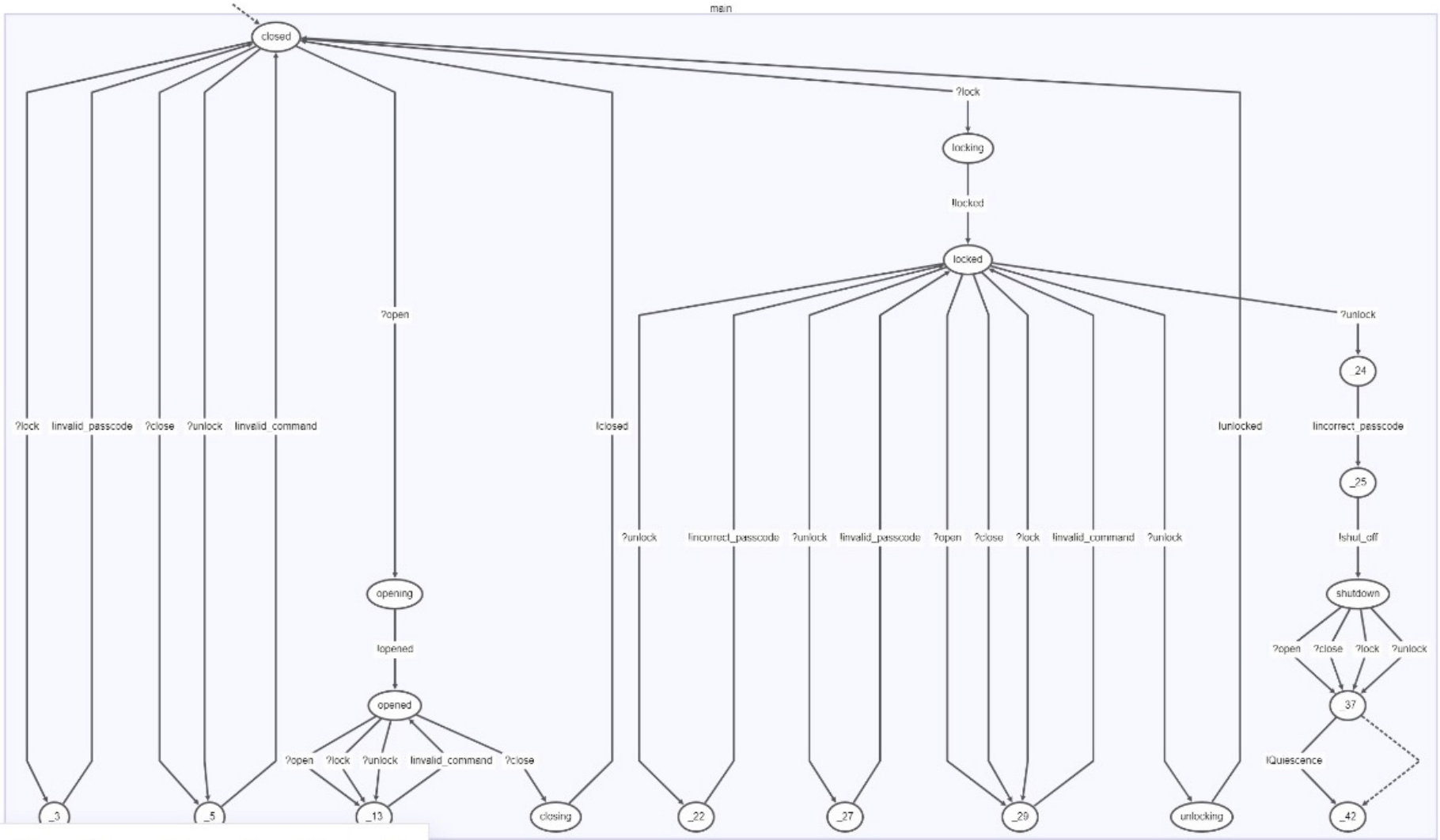
Process mining

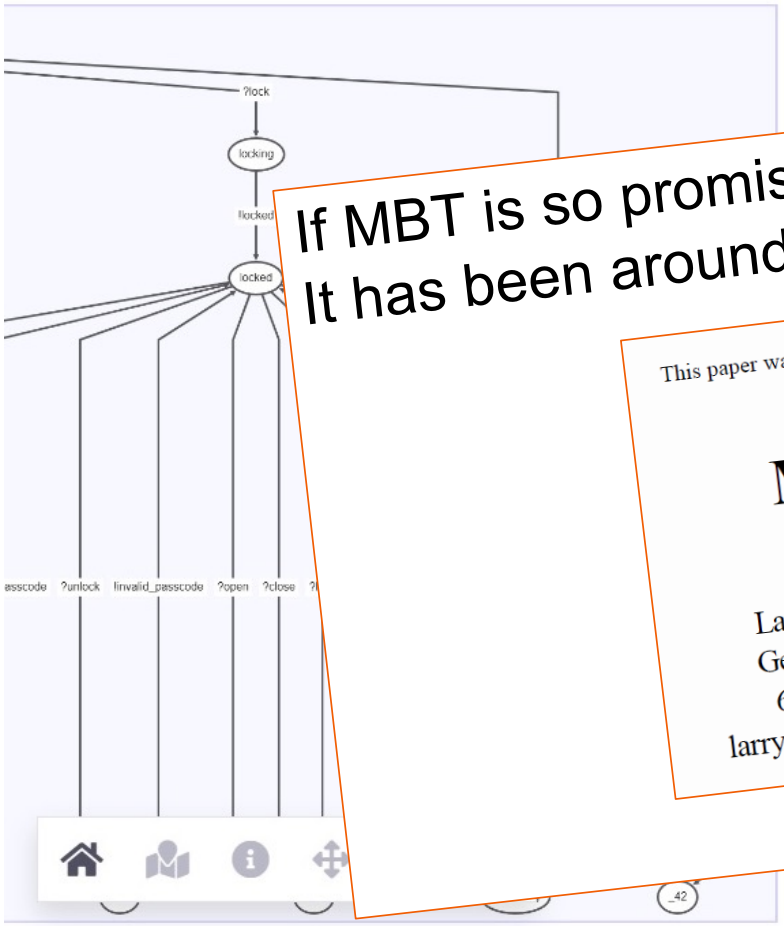


Development *Test*

Process mining




main
closed
?lock
locking
!locked
locked
?unlock
24
!incorrect_passcode
_25
!shut_off
shutdown
?open
?close
?lock
?unlock
37
!Quiescence
_42
?lock
!invalid_passcode
?close
?unlock
!invalid_command
_3
_5
?open
opening
!opened
opened
?open
?lock
?unlock
!invalid_command
?close
_13
!closed
closing
?unlock
!incorrect_passcode
?unlock
!invalid_passcode
?open
?close
?lock
!invalid_command
?unlock
_22
_27
_29
!unlocked
unlocking



Source: smartdoor.aml

```
63    state 'locked'
64    choice {
65      o {receive 'unlock', constraint: '                    '; goto 'unlocking'}
66      # check '
                                                  passcode <= 9999 &&
                                                  fails = fails + 1';

                                                  passcode <= 9999 &&
                                                  off', constraint:

                                                  scode > 9999';

                                                  ed'}
                                                  ed'}
                                                  ode <= 9999';
80
81    state 'shutdown'
82    choice {
83      o  receive 'open'
```

If MBT is so promising, why is not everyone using it?
It has been around for >25 years

This paper was distributed at the Software Quality Week Conference in May, 1997.

Model Based Testing

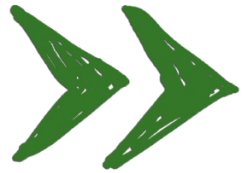
Larry Apfelbaum
General Manager
603-879-3555
larry@sst.teradyne.com

John Doyle
Support Manager
603-879-3499
jd@sst.teradyne.com

Challenges of MBT - Complexity

- Stakeholders have different expectations of MBT
  - Shorter leadtime vs. higher quality
- Modeling is a specialized skill
  - Some testers find coding hard… modeling can be even harder
- Not every (part of a) system is suited for modeling
- Tooling landscape seems big… but:
  - Mostly GUI tools... Avoid automated testing via GUI
  - Most tools do not support non-determinism / uncertainty

Vision of SIOUX TECHNOLOGIES



TorXakis

axini

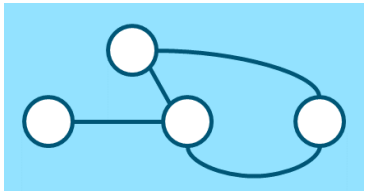
Founded in ioco-testing theory

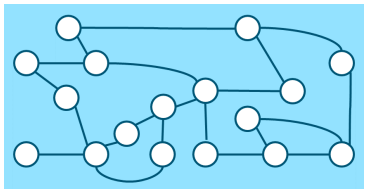
References:

- https://www.axini.com/en/products/model-based-testing/
- https://github.com/TorXakis/TorXakis

Challenges of MBT – State space explosion

- Expectations vs reality
  - Abstraction level of models
    Unclear scope of models leads to wrong abstraction level of models:
    - too abstract : model has limited to no added value
    - too detailed : high costs, state space explosion





  - MBT applied on too many areas → high costs, disappointing benefit
    - Apply only for high-risk areas
    - Focus on e.g. performance, reliability

Challenges of MBT – State space explosion

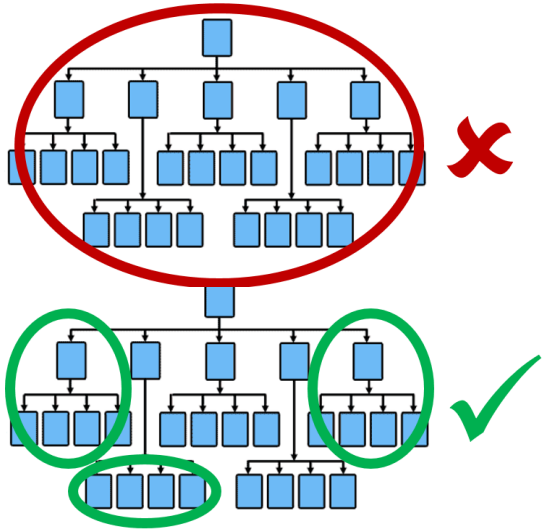
Dealing with state space explosion

Scope Focus on **components / subsystems**

instead of the **entire system**

a) simple models for different test purposes

b) based on risk analysis



Abstraction Focus on **behavior**, instead of **design**

a) limit number of data values (use S, M, L iso range 0-1000)

b) model the behavior instead of the design

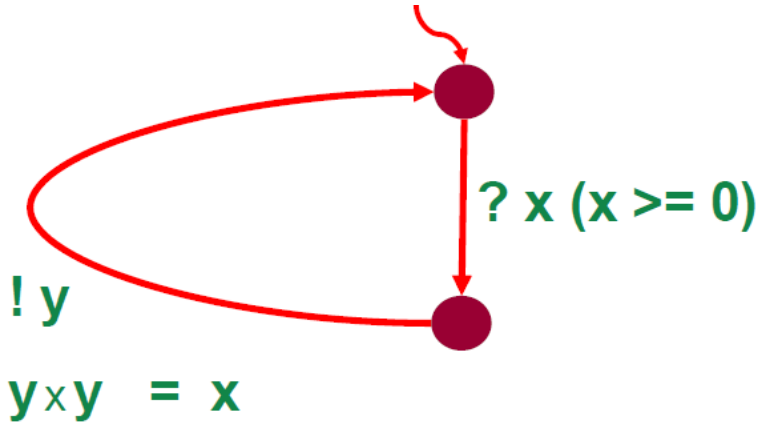
Reference:

- Groote, Kouters, Osaiweran. *Specification guidelines to avoid the state space explosion problem*

Challenges of MBT – State space explosion

Dealing with state space explosion

- model the behavior instead of the design

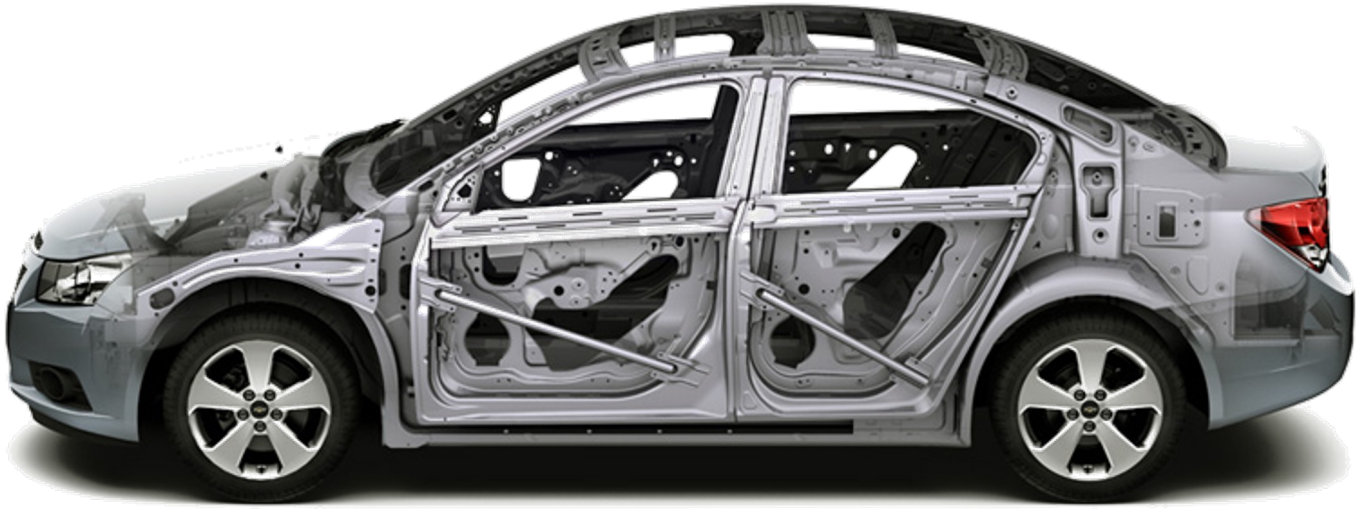


model of $\sqrt{x}$

Reference:
- Jan Tretmans – Radboud University Nijmegen – Model Based Testing

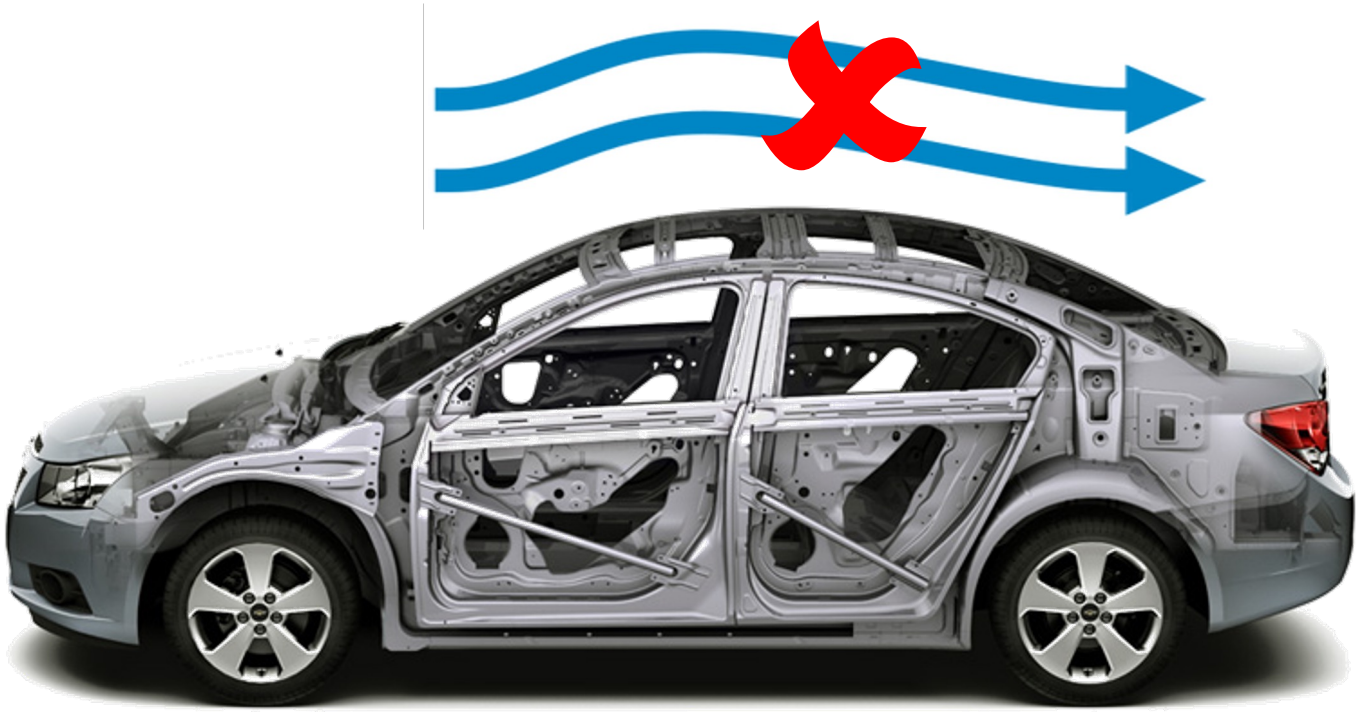
Challenges of MBT – State space explosion

Different models for different purposes



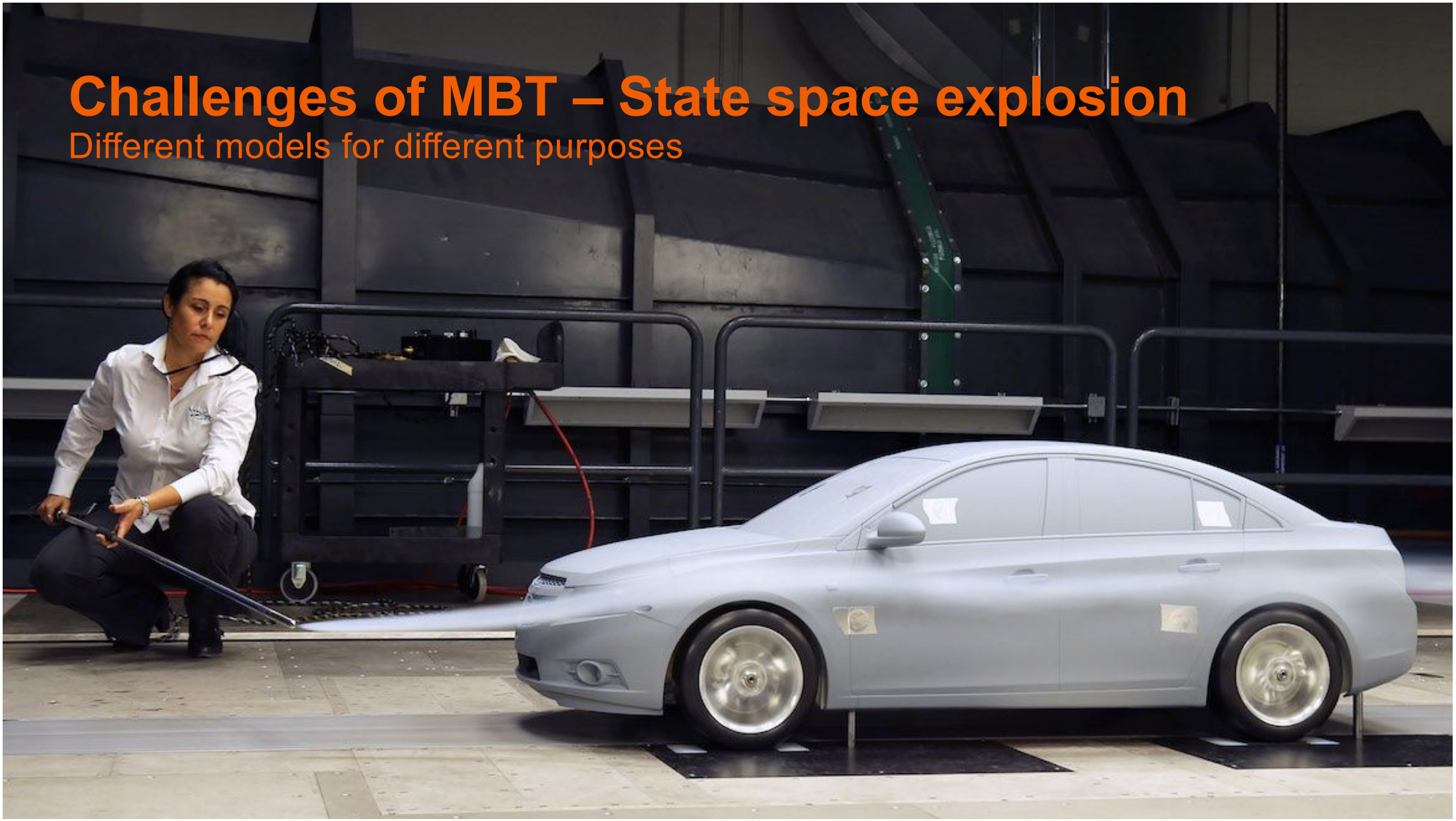
Challenges of MBT – State space explosion

Different models for different purposes



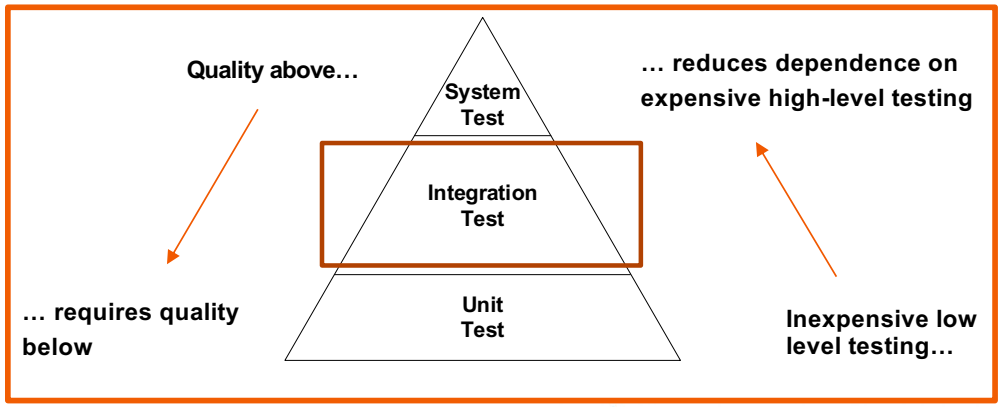
Challenges of MBT – State space explosion

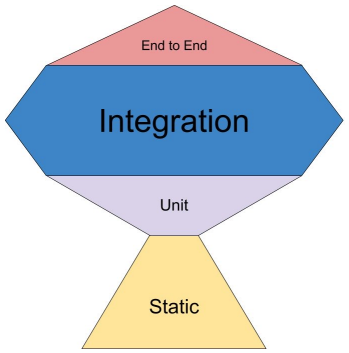
Different models for different purposes

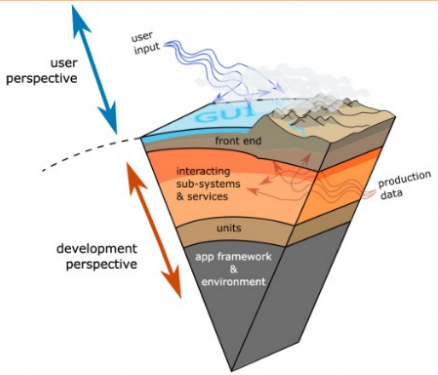


MBT helps to align levels in test pyramid

- Don’t include details in the model that are (or should be) covered in lower levels → abstract models
- MBT does not replace unit tests
- Applying MBT on integration level
- Do not repeat checks → will “relieve” system level
- Minimize number of testcases on upper levels
  - Expensive to define, even more expensive to maintain
  - Slow execution
  - Complex / expensive / scarce environments
  - Hard analysis of failures

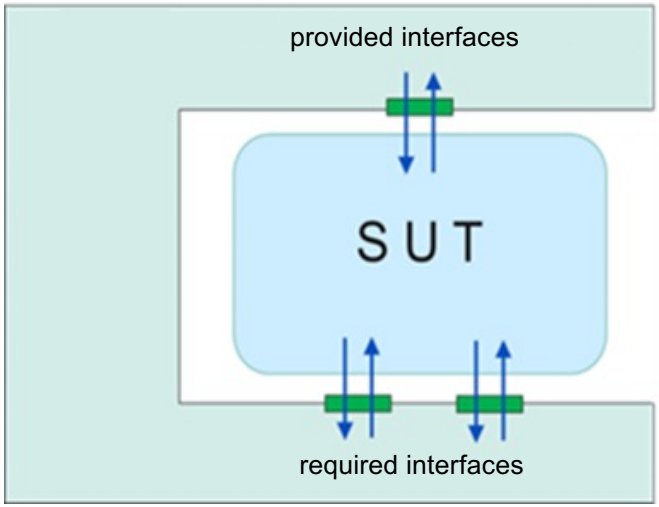






References:
- Round Earth Test Strategy - Satisfice, Inc.
- Test trophy - Kent C. Dodds

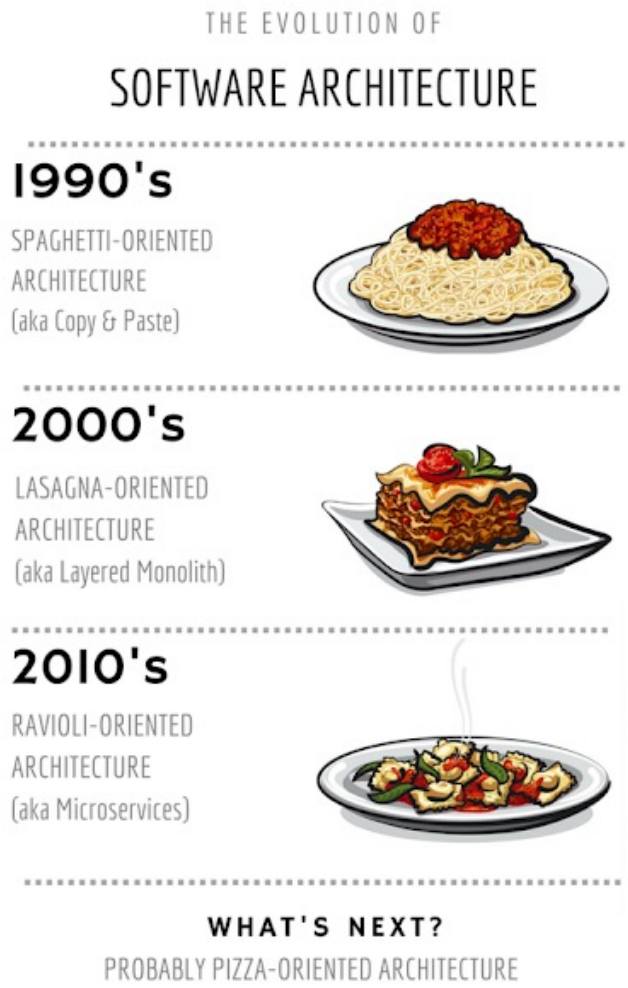
Experience



- Modeling both provided AND required interfaces enables automated testing of both happy and non-happy flow
- MBT platform generates a lot of test variations, not straightforward with BDD approach
- Modeling gives you early feedback on interfaces and requirements (even before implementation) → might be perceived as delay
- Projects using MBT have less issues in later test phases
- Test sub-systems in isolation without the need for simulators
- Load on testbenches and proto's is reduced

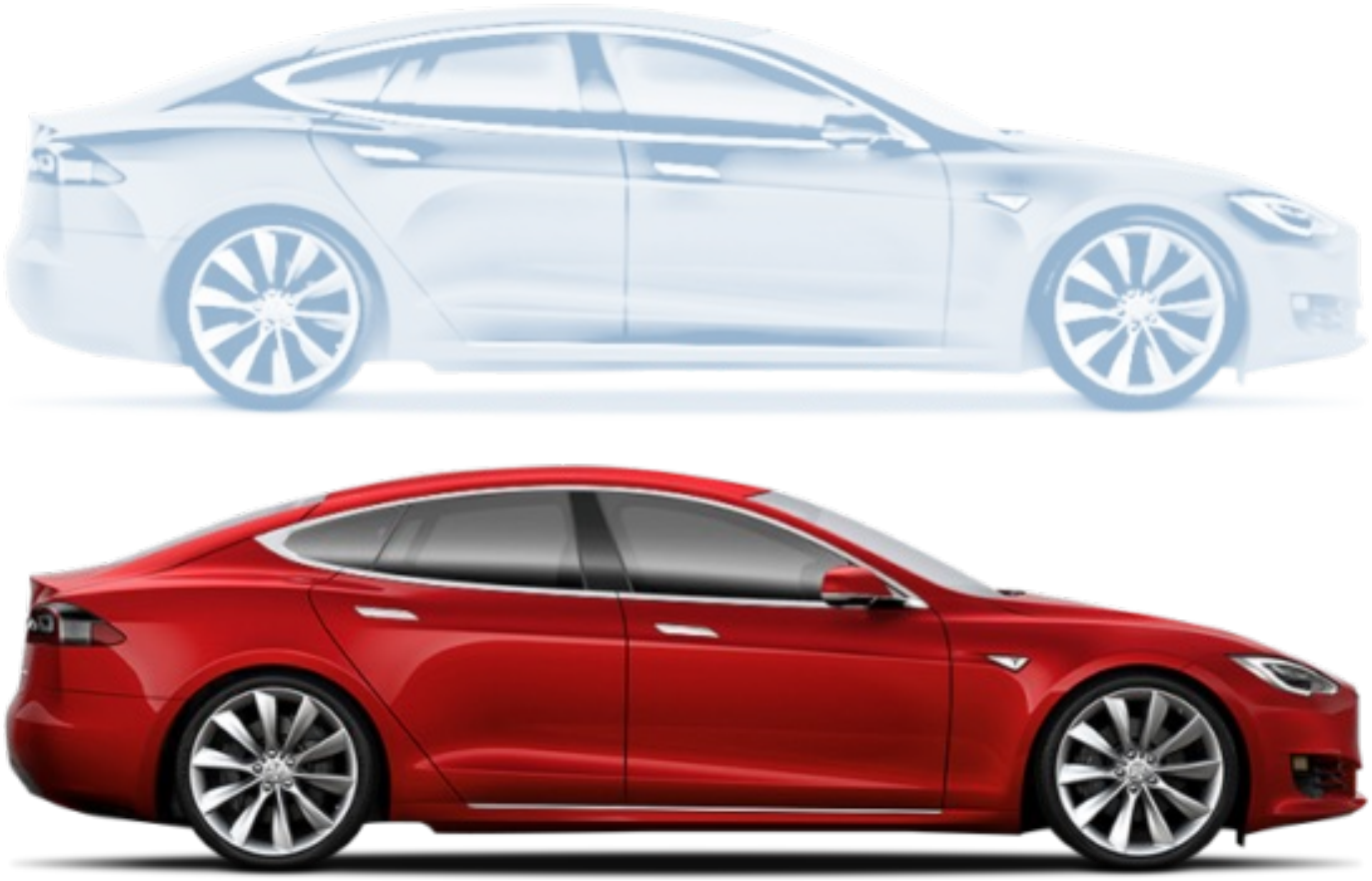
Experience

- Opportunities:
  - Green field
  - Bigger redesigns / refactoring
  - Interface migrations
  - Increase test coverage
  - Requirements elicitation
- Experience shows that teams who model:
  - Deliver higher quality
  - Have higher productivity

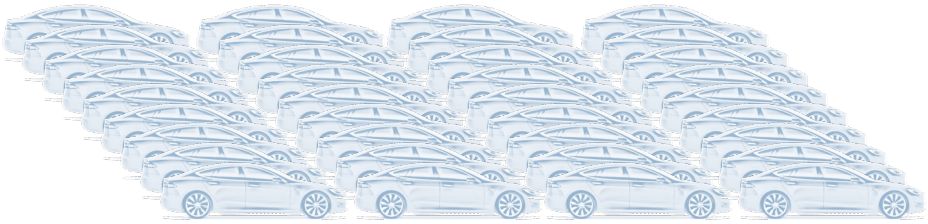


Promises for the future

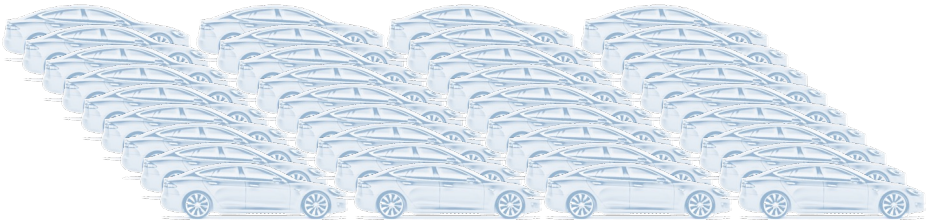
Promises for the future – Digital twins

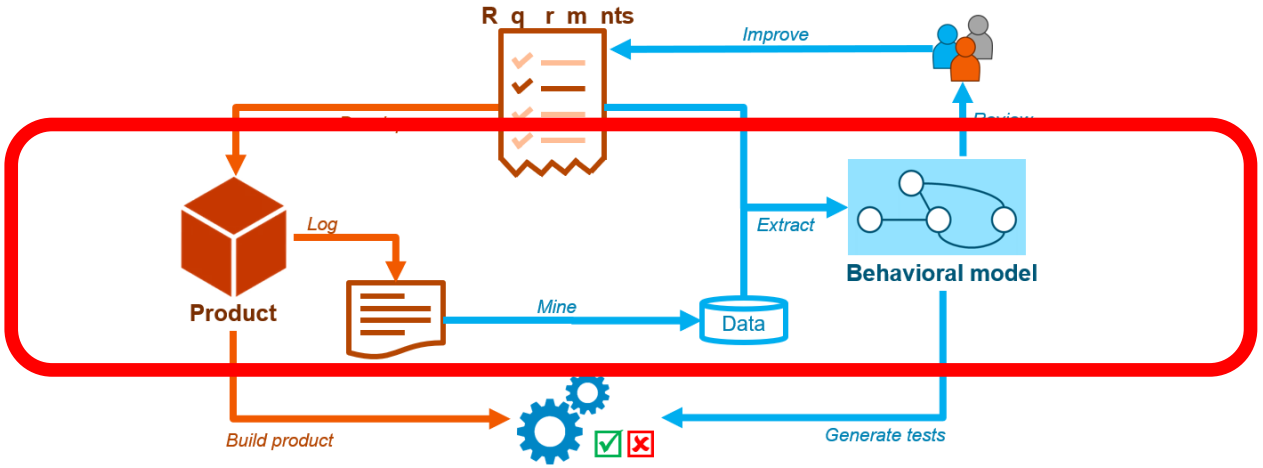


Promises for the future – Digital twins

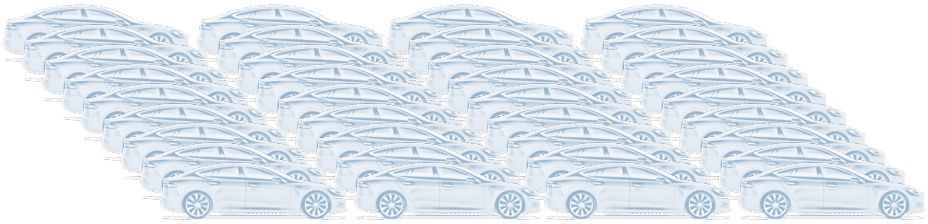


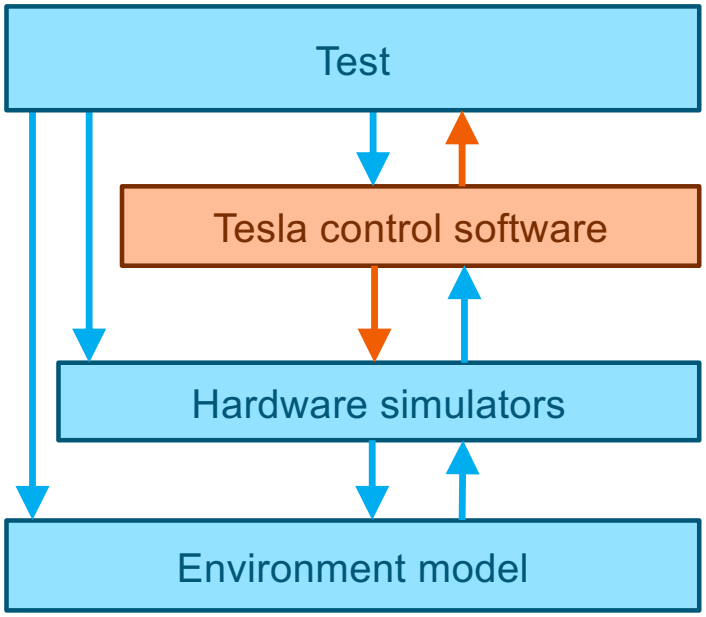
Promises for the future – Digital twins



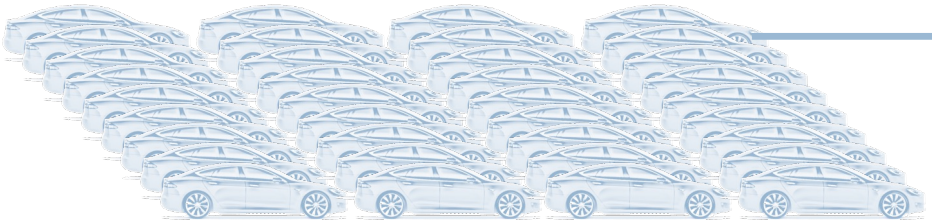


Promises for the future – Digital twins

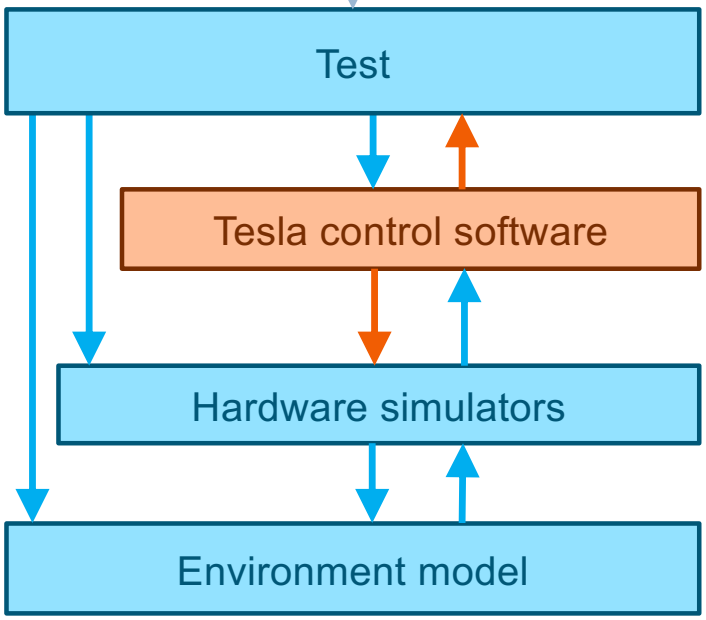




Promises for the future – Digital twins



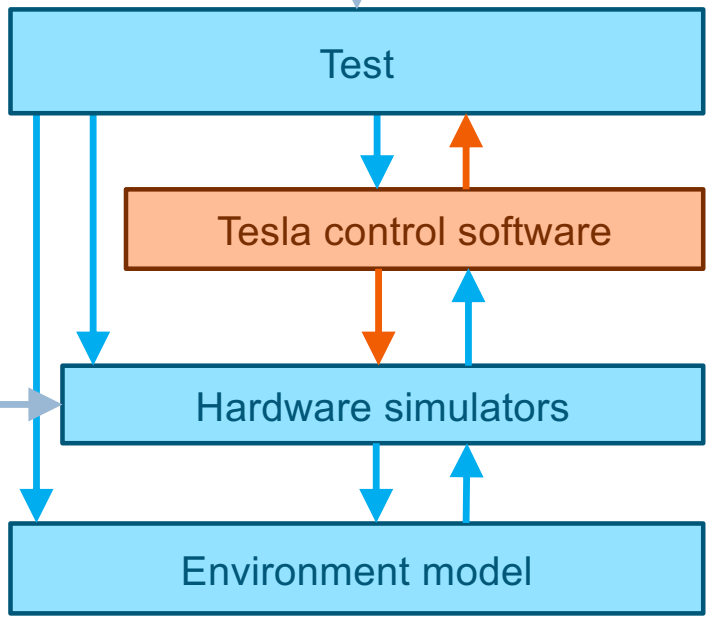
- *Increase test coverage*
- *Identify operational profiles*
- *Statistics for risk analysis*
- *Identify behavior models*



Promises for the future – Digital twins

- *Increase test coverage*
- *Identify operational profiles*
- *Statistics for risk analysis*
- *Identify behavior models*

Improve simulators based on
- *Influence of wear and tear*
- *Environment circumstances*



Promises for the future – Digital twins

- *Increase test coverage*
- *Identify operational profiles*
- *Statistics for risk analysis*
- *Identify behavior models*

Improve simulators based on
- *Influence of wear and tear*
- *Environment circumstances*

Increase environment variations

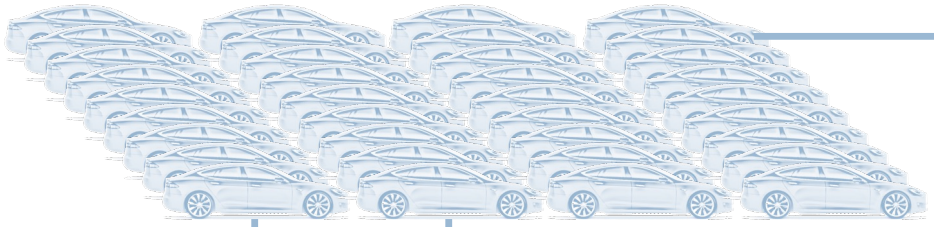
Test

Tesla control software

Hardware simulators

Environment model

Promises for the future – Digital twins



- *Increase test coverage*
- *Identify operational profiles*
- *Statistics for risk analysis*
- *Identify behavior models*

FIL: Field in the loop testing

- *Improve simulators based on*
- *Influence of wear and tear*
- *Environment circumstances*

Increase environment variations

Test

Tesla control software

Hardware simulators

Environment model

Future? - Iterating in virtual space

- Growing use of digital twins
- Fully virtual target environment
  - Including autonomous driving
  - Other cars (V2V)
  - Weather conditions
  - Pedestrians / bikes
  - V2I communication
  - … V2X
- Thousands of hours of driving, tested within seconds in CI/CD cycle

→ Modeling the known

The Magic Roundabout, Swindon, England

Future? – Use of machine learning

- Autonomous driving should be able to handle the unknown
- Improve virtual environment with data mining
  - Actual info from the field
  - E.g. Pedestrians / other cars not behaving as expected
  - Identifying new unknown scenarios

→ Modeling the unkown (and make it known)

Future? – Soon, this is not needed anymore…?

Erlkönig (Camouflaged prototype)

Questions?

bryan.bakker@sioux.eu
dirk.coppelmans@sioux.eu

The Magic Roundabout, Swindon, England