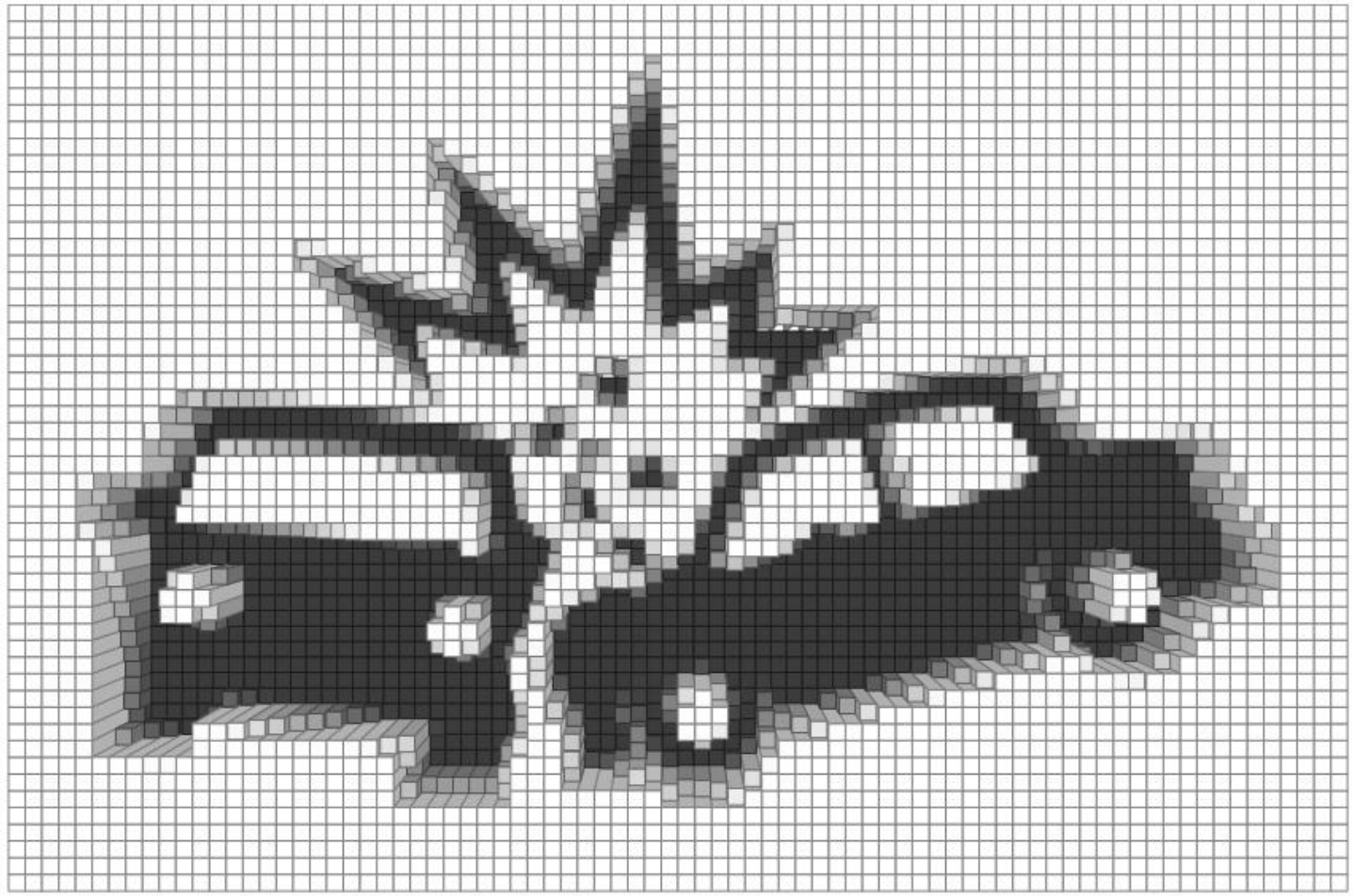
The future of models in testing

Safely crash in virtual space

SIOUX TECHNOLOGIES



Bryan Bakker

Dirk Coppelmans

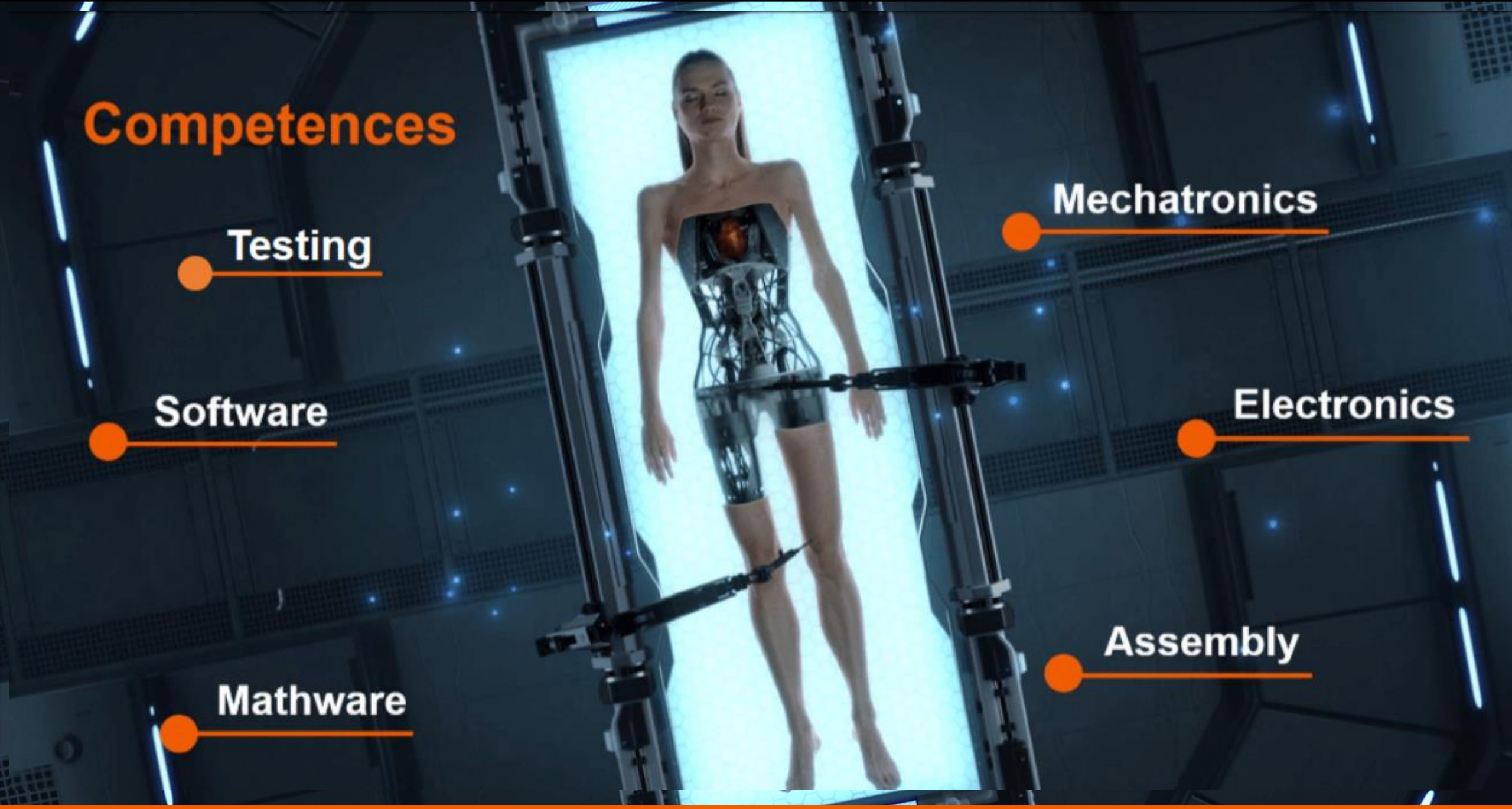
Bryan Bakker

- 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



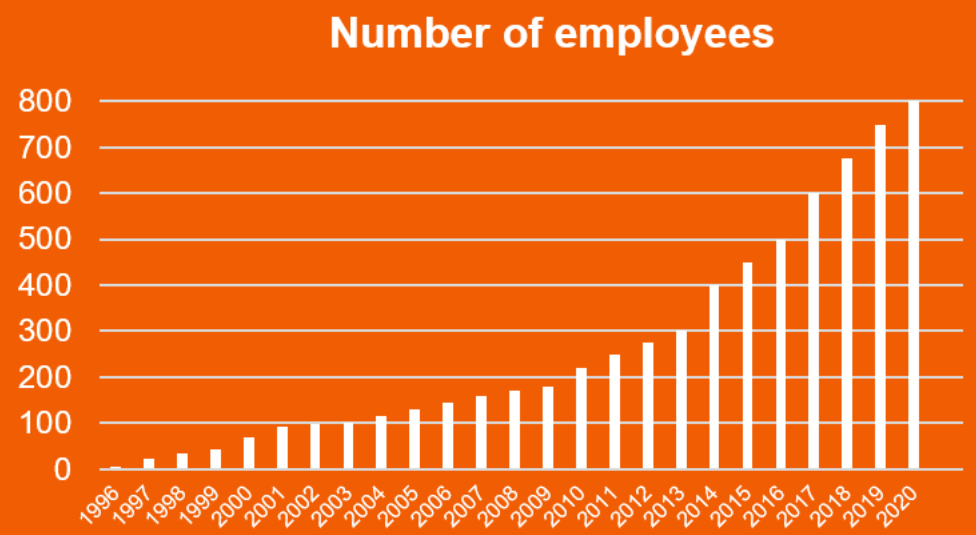
The facts

Annual turnover
€ 80,000,000

Customer satisfaction
8

Technical staff
700+

Employee satisfaction
8



The future of models in testing

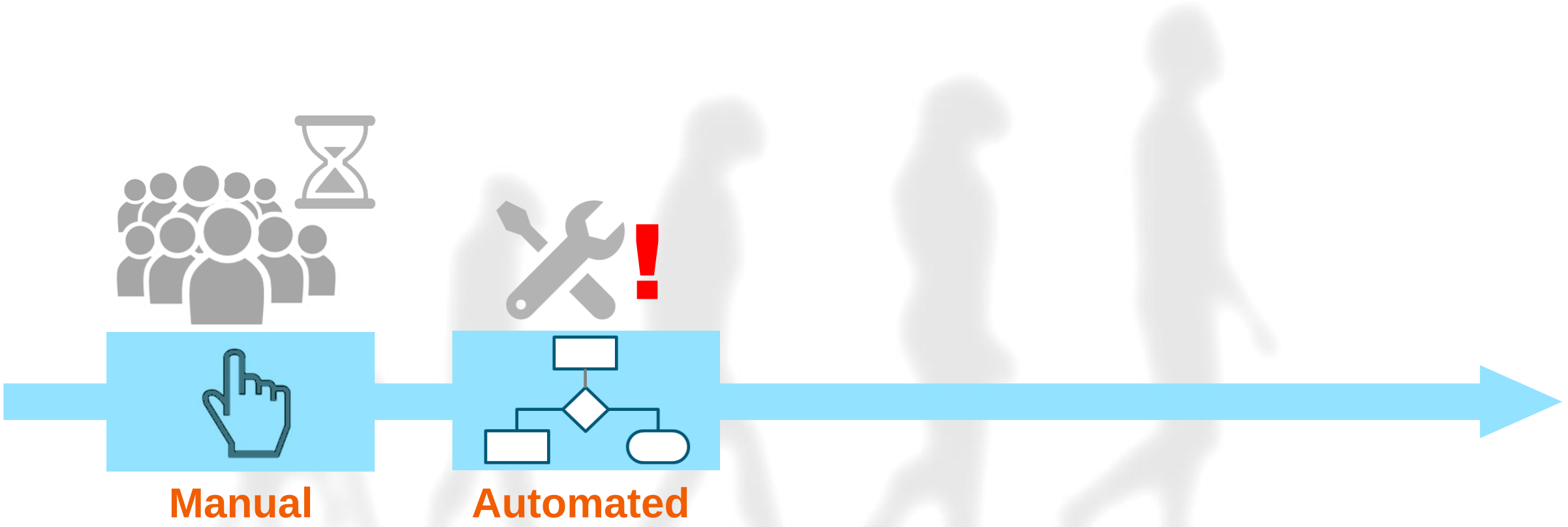
1. Evolution of software testing
2. Models in testing
3. Future?

Evolution of software testing

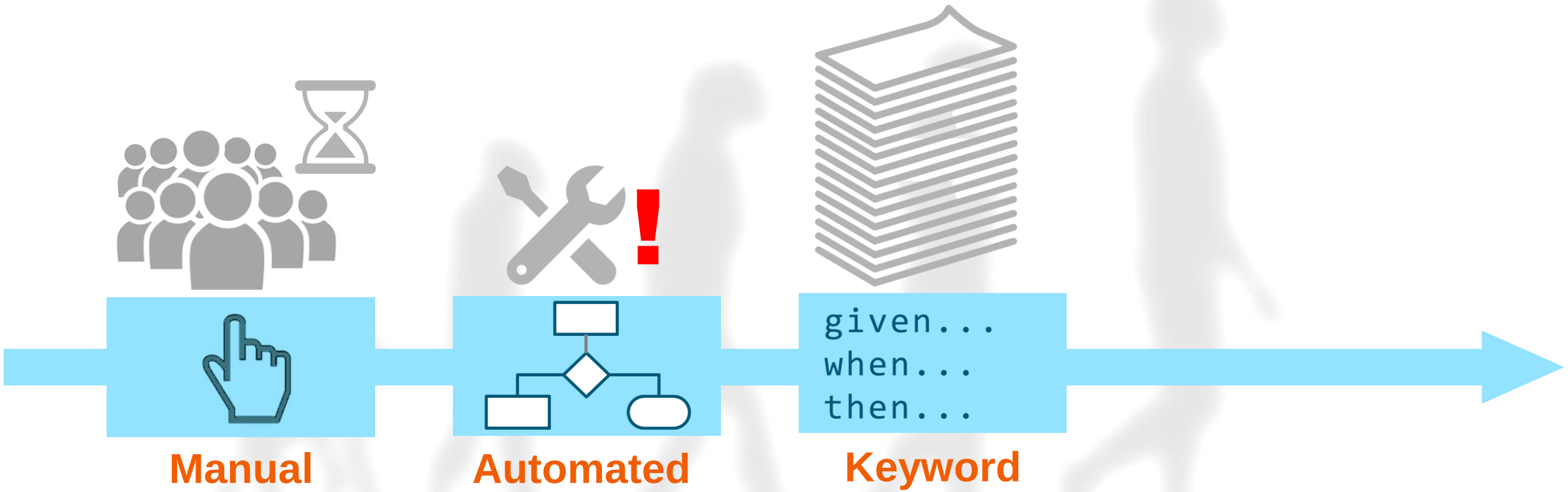
Evolution of software testing

Evolution of software testing

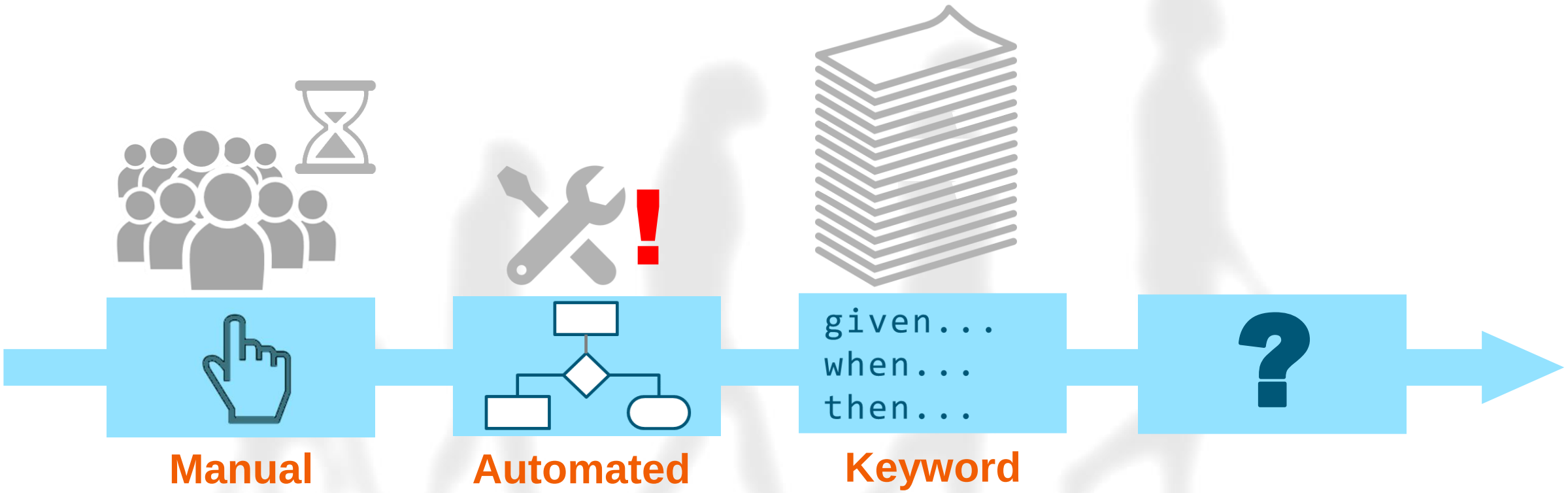
Evolution of software testing



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

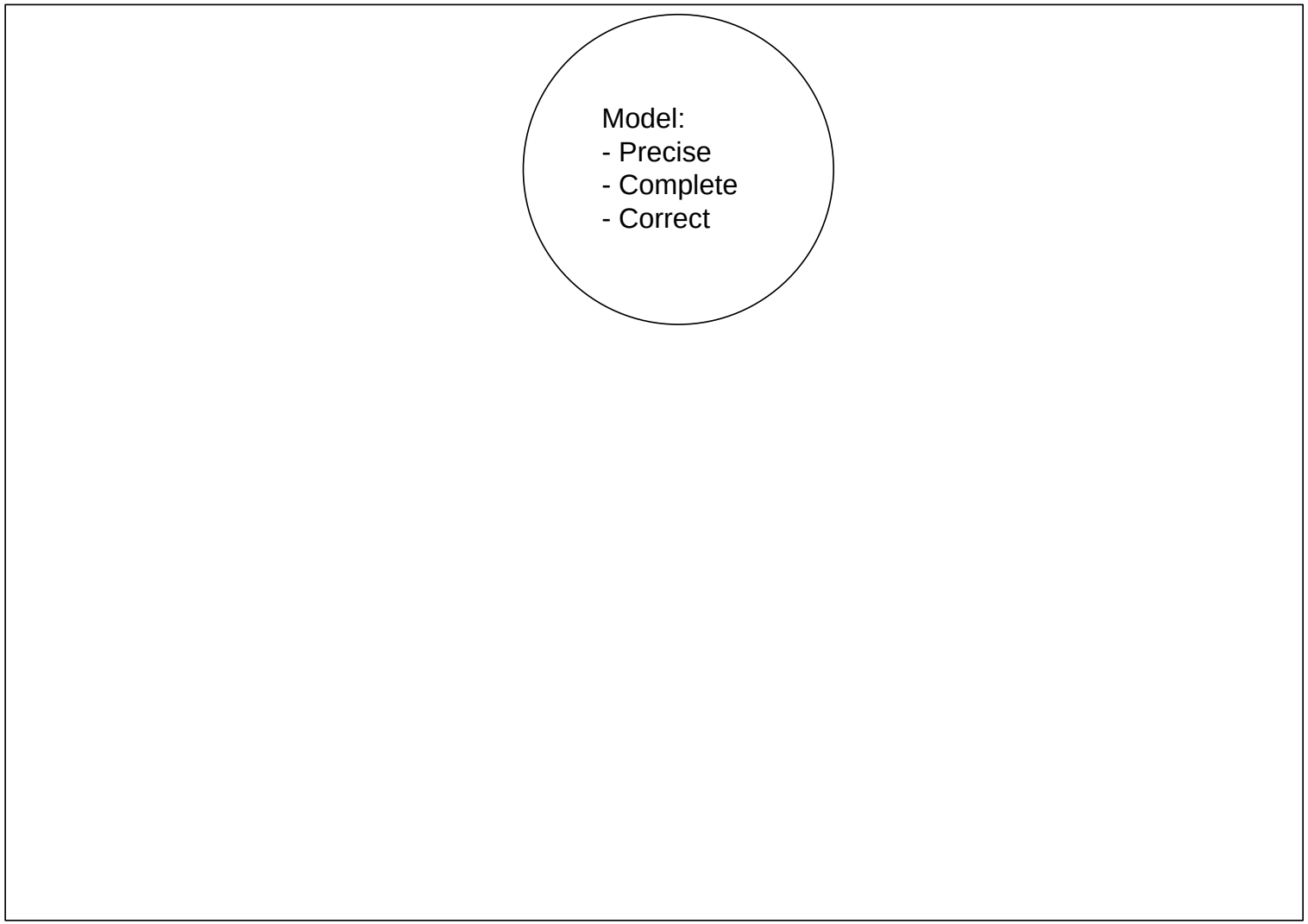
Type	Tools
System design	• ASD / Dezyne (Verum) • mCRL2 (verification engine for ASD/Dezyne) • Cocotec
Interface	• Pact • ComMA

verum.

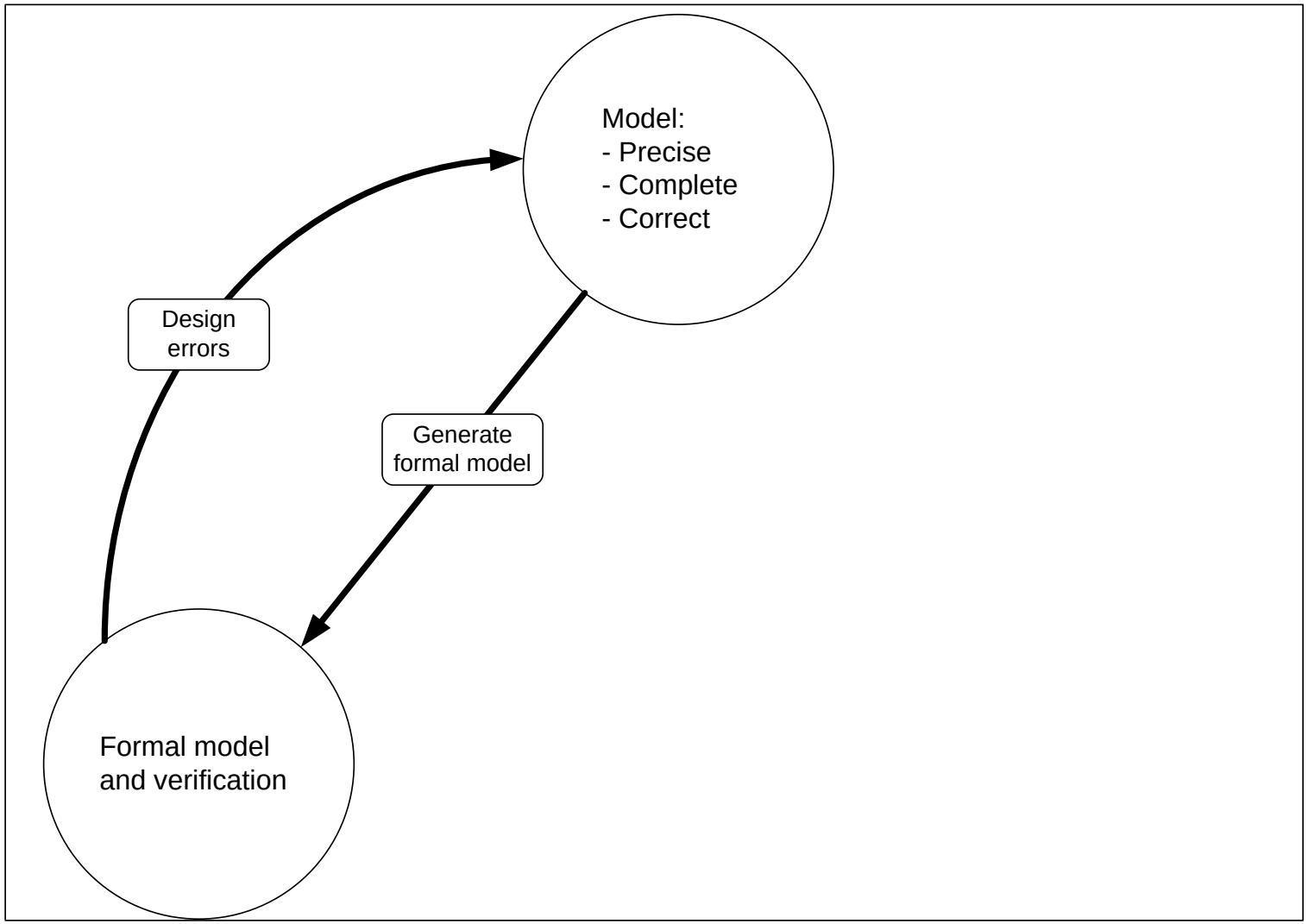
COCOTEC

PACTS

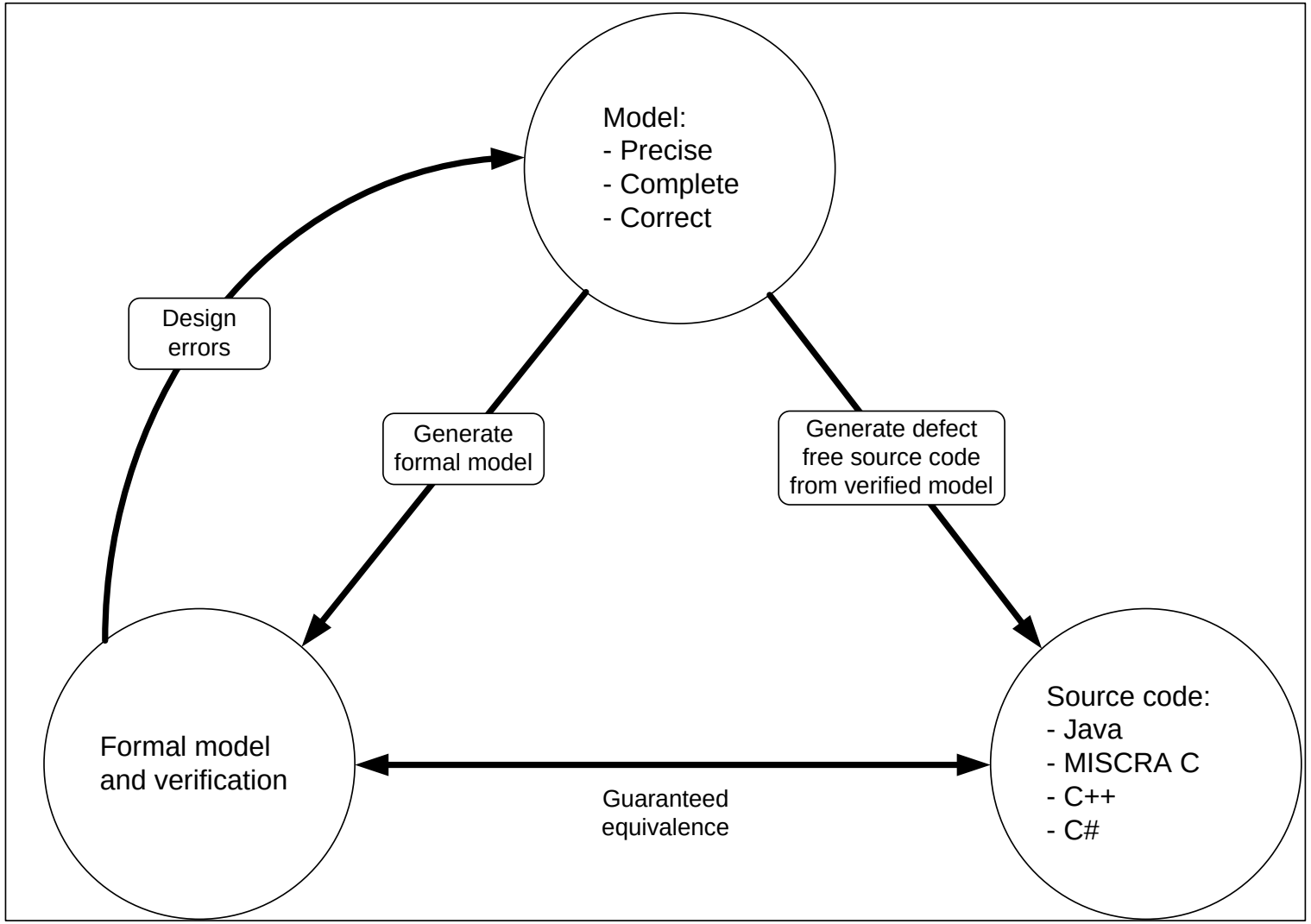
Model Driven Development (MDD) - Model



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

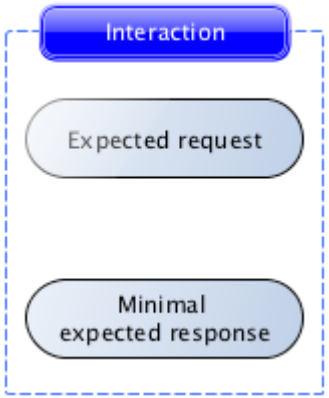
Models in testing

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, Schematesis

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

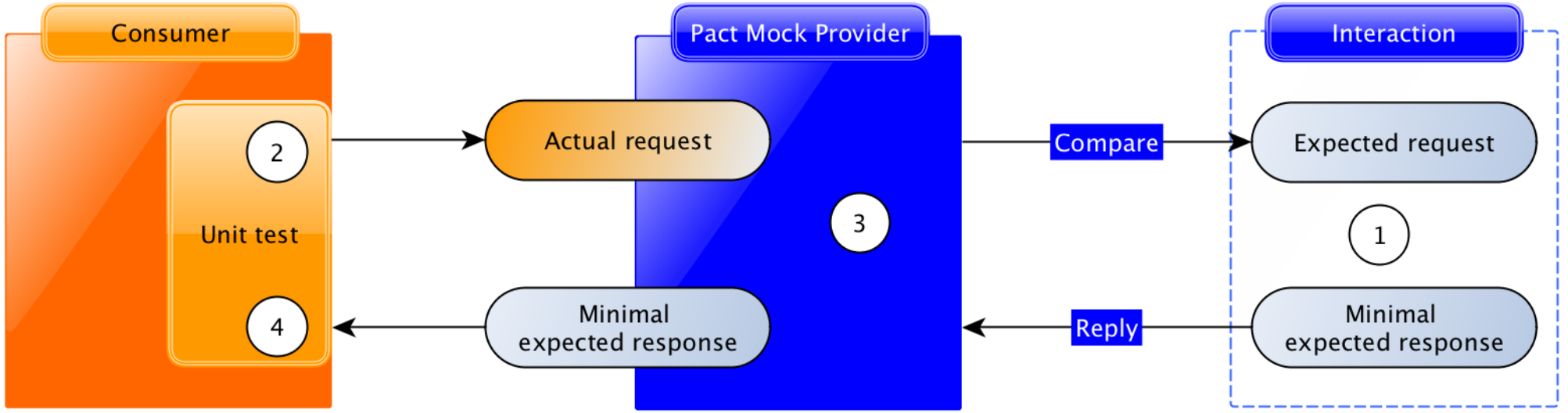
Contract



- **Consumer**: A client that wants to receive some data (for example, a web front end, or a message receiving endpoint).
- **Provider**: A service or server that provides the data (for example, an API on a server that provides the data the client needs, or the service that sends messages).
- A **contract** between a consumer and provider is called a pact. Contract are consumer driven. Each pact is a collection of interactions. Each interaction describes:
  - An expected request - describing what the consumer is expected to send to the provider
  - a minimal expected response - describing the parts of the response the consumer wants the provider to return.

https://pact.io/

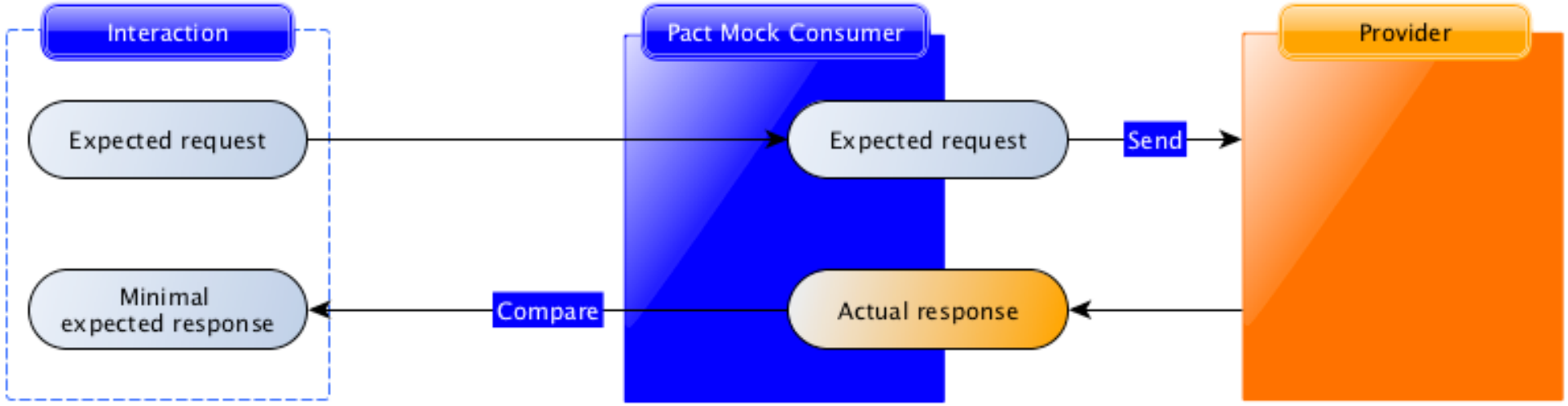
Consumer side testing



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/

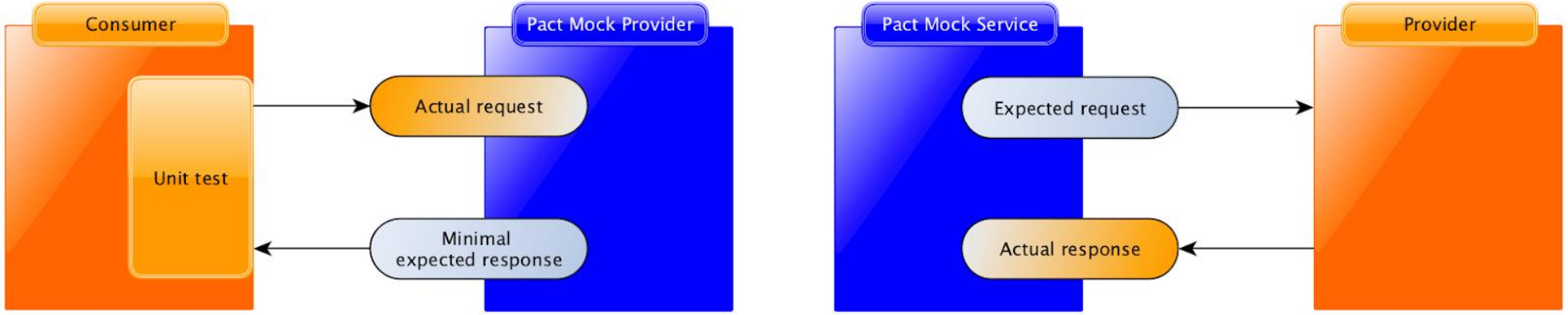
Provider side testing



- 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

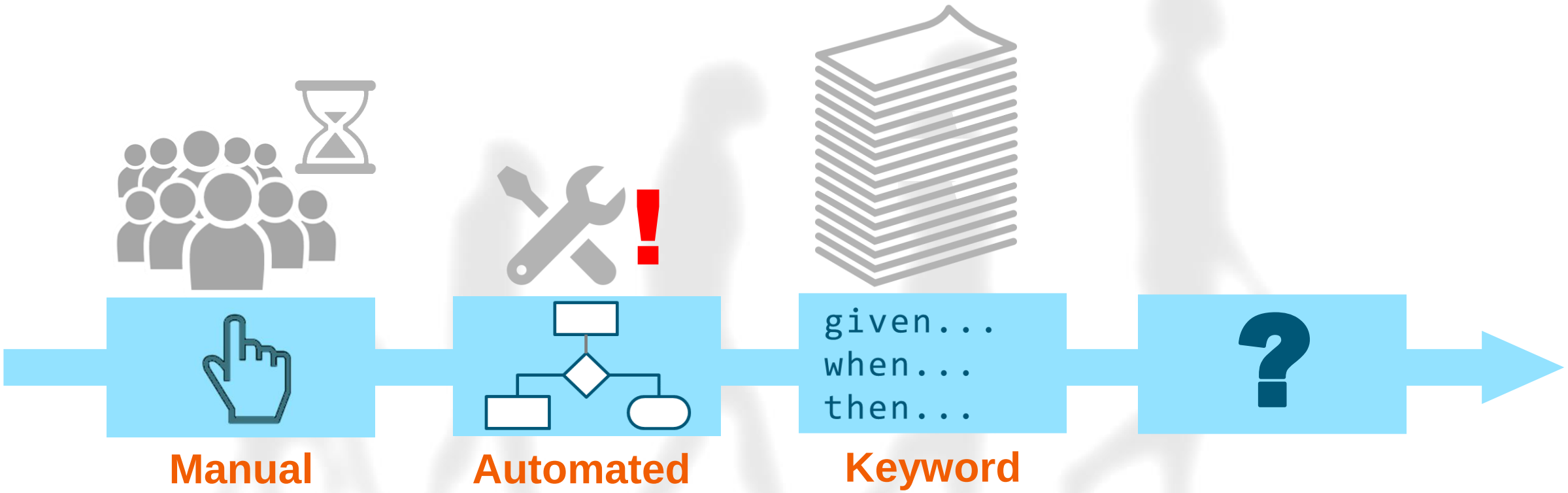
References:
- https://projects.eclipse.org/projects/technology.comma
- https://esi.nl/research/output/tools/comma

Modeling

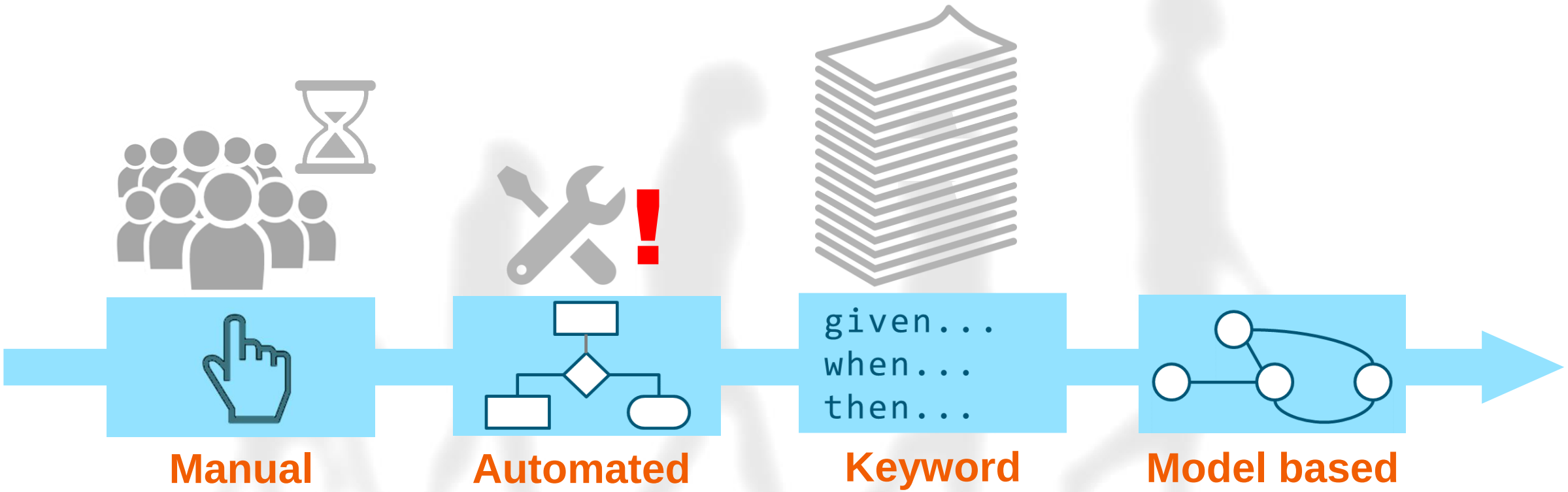
- MDD Design models are by far flawless
  - Often only complex+critical parts of the system modeled
- Interface models are
  - Rigorous
  - Valuable for interface evolution
  - Limited to interfaces

- By modeling behavior, new test possibilities arise

Evolution of software testing



Evolution of software testing



MBT Definition

Model based testing is

automated test generation and execution

based on an abstract

behavior model

Developing a behavior model

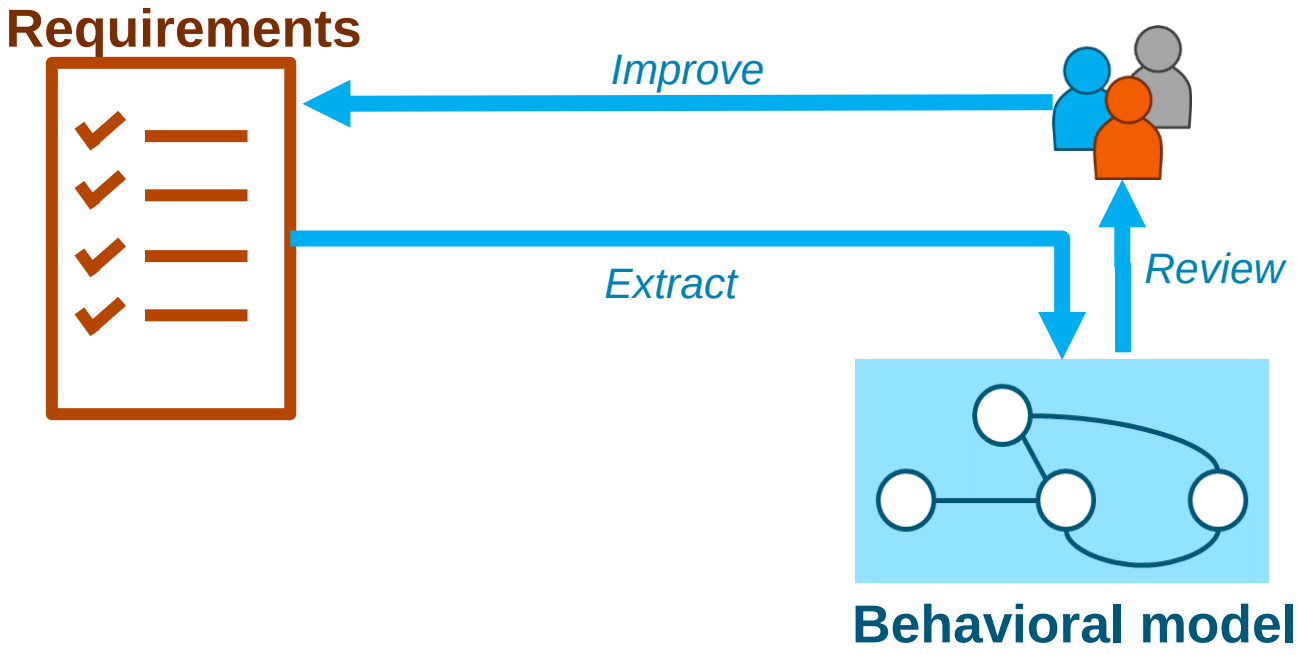
1. Derive the behavior model from the requirements
2. Construct a behavior model based on collected field data (process mining)

Derive from the requirements

Development

Test

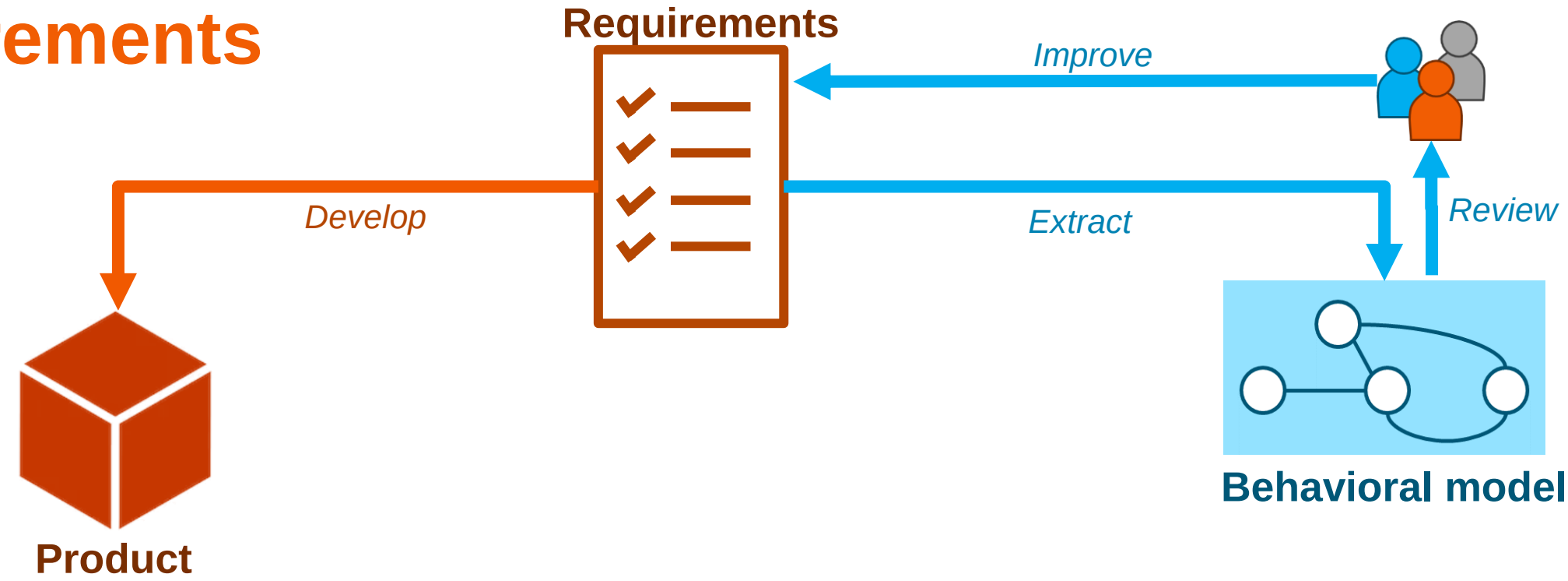
Derive from the requirements



Development

Test

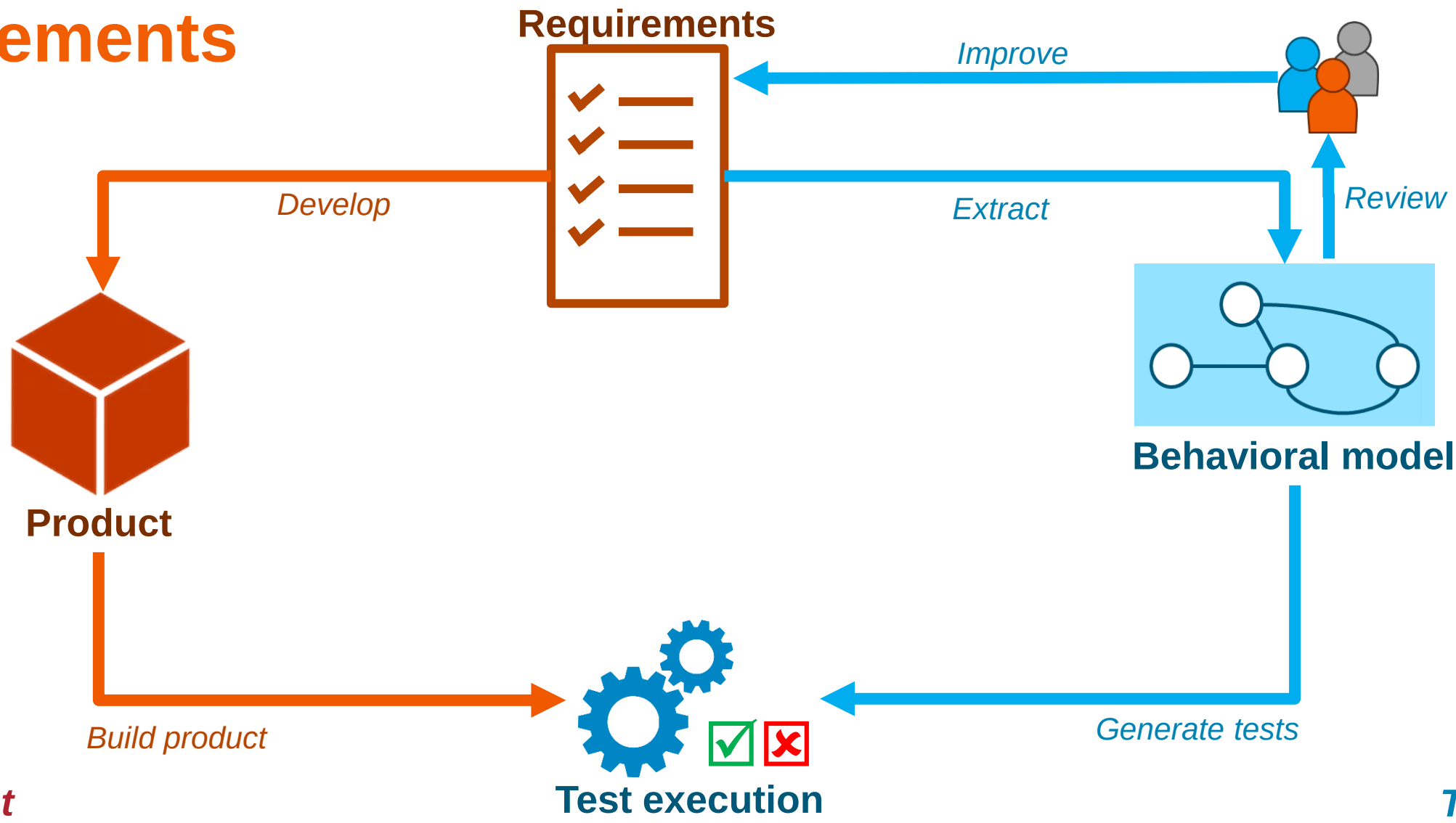
Derive from the requirements



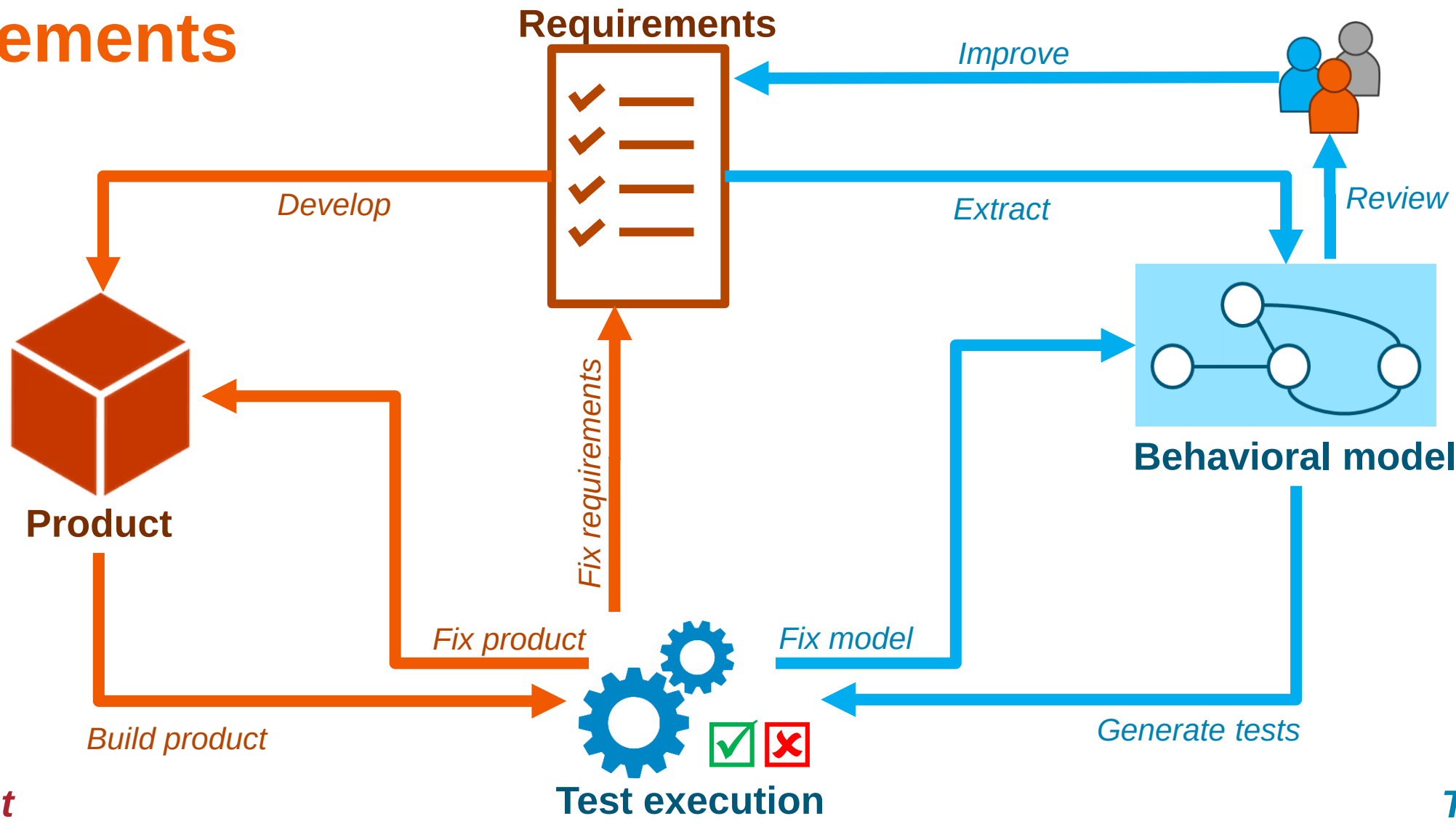
Development

Test

Derive from the requirements



Derive from the requirements

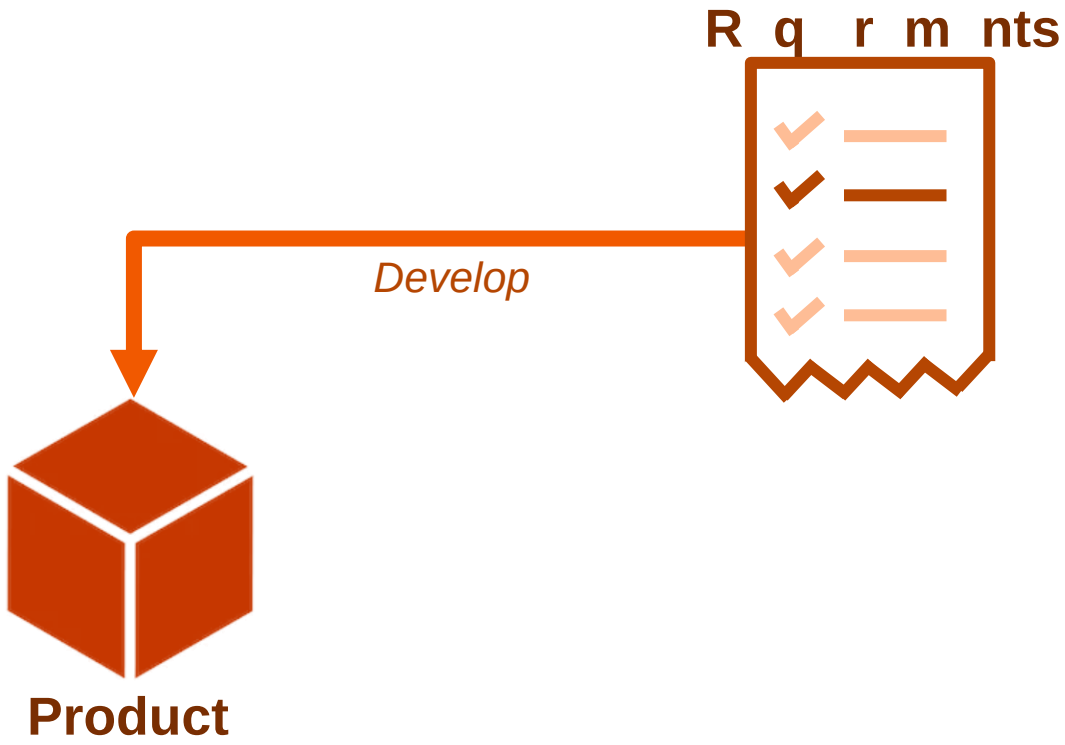


The next best thing – Developing a behavior model

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

Process mining

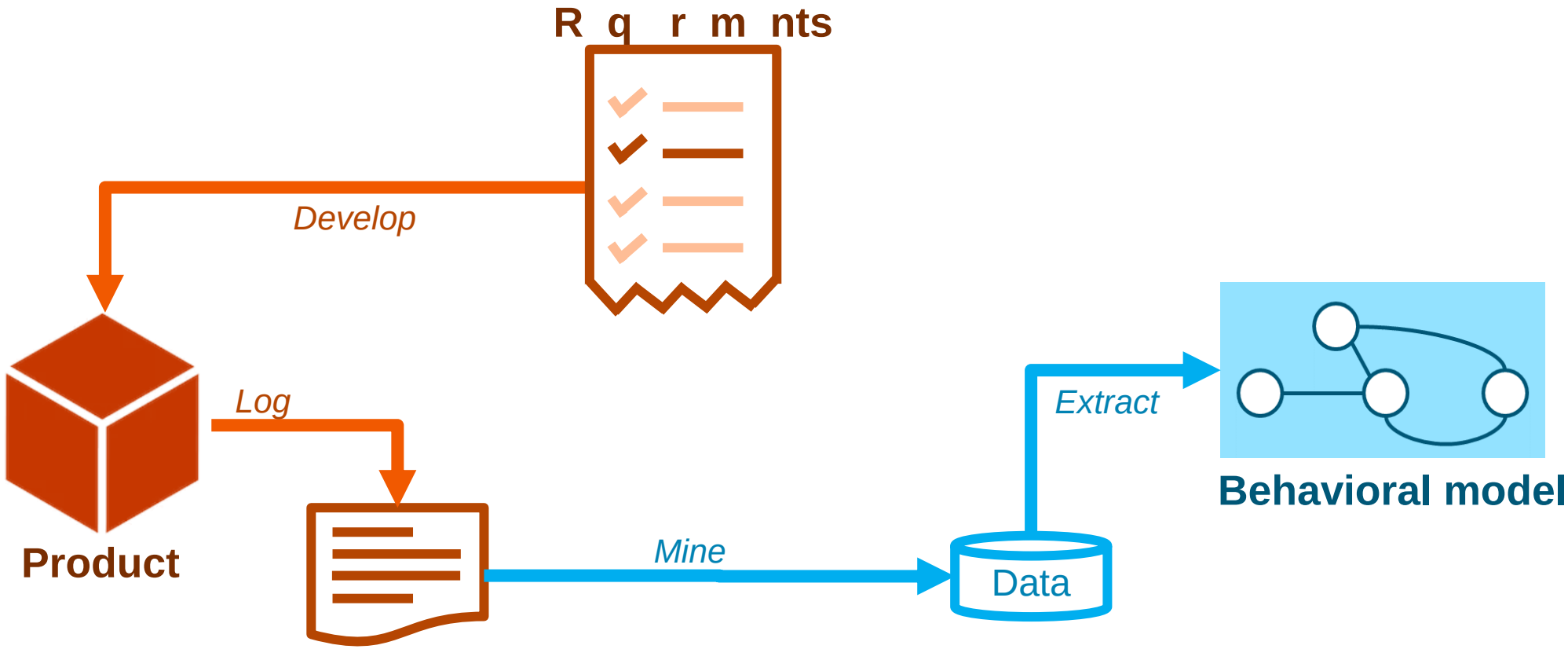
Process mining



Development

Test

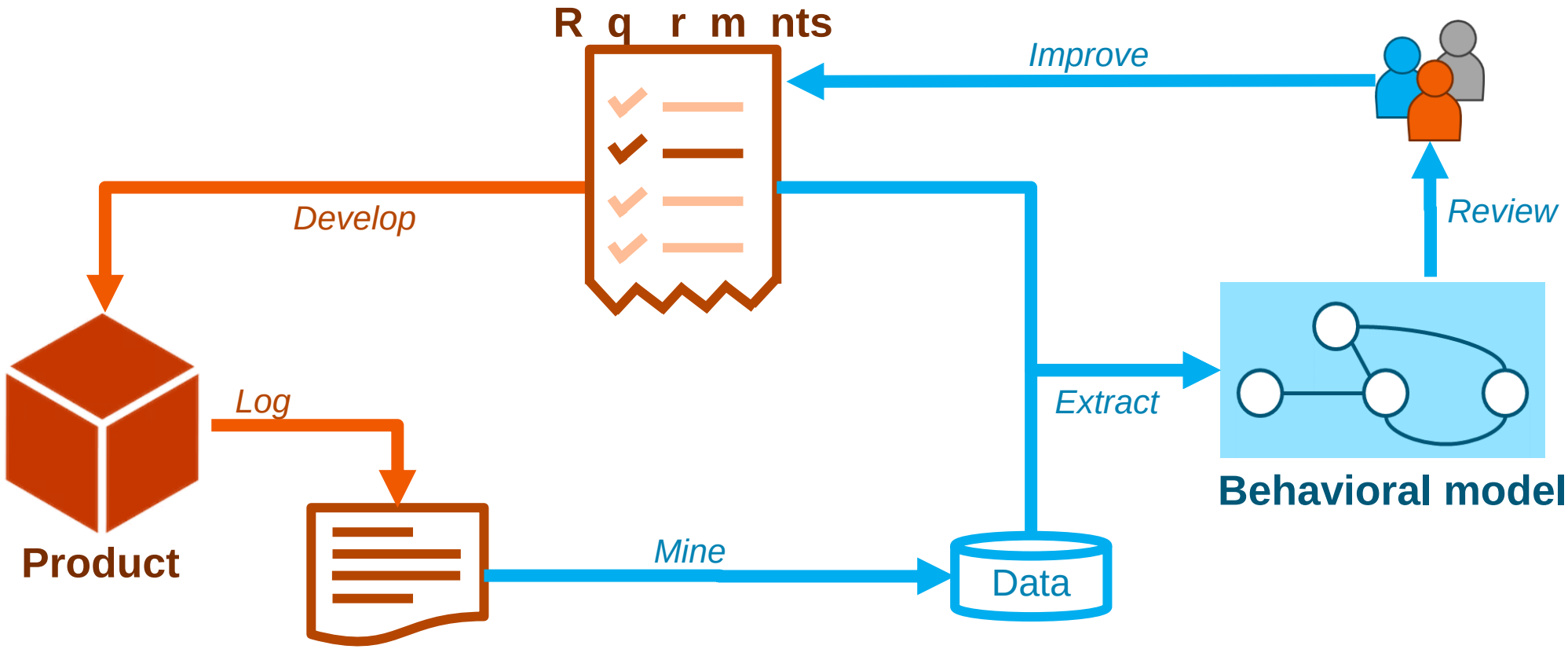
Process mining



Development

Test

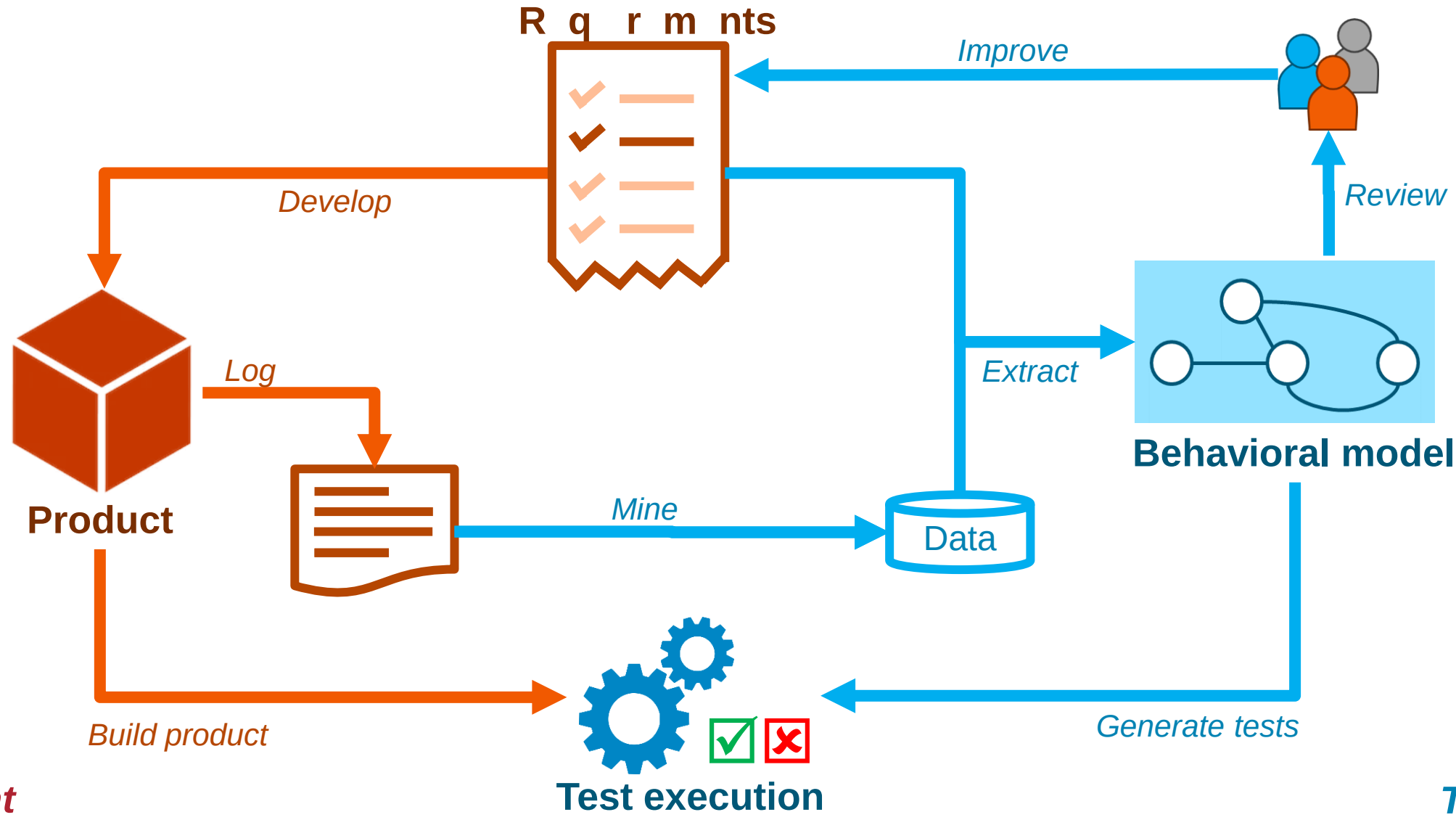
Process mining



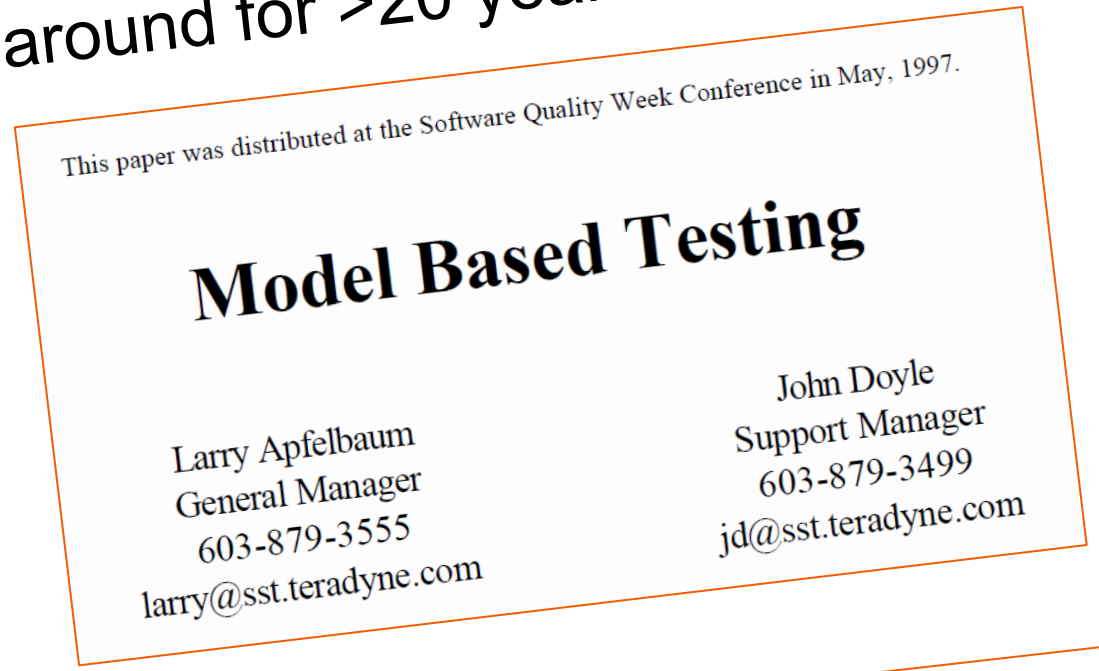
Development

Test

Process mining



If MBT is so promising, why is not everyone using it?
It has been around for >20 years



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

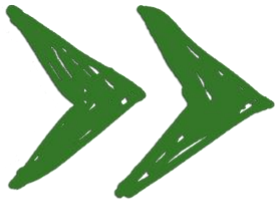
Model Based Testing

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
- There is a lack of mature tools
  - Mostly GUI tools... Avoid automated testing via GUI
  - Most tools do not support non-determinism / uncertainty



TorXakis

axini

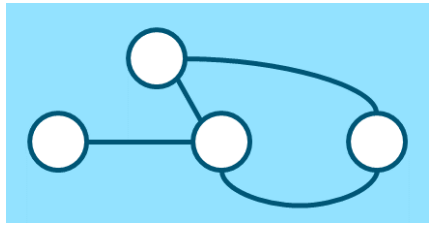
Founded in ioco-testing theory

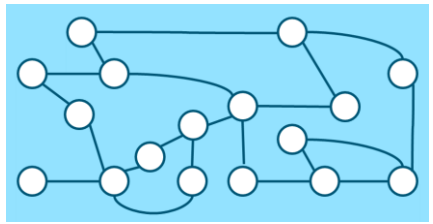
References:

- 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
    - Performance, reliability & security

Challenges of MBT – State space explosion

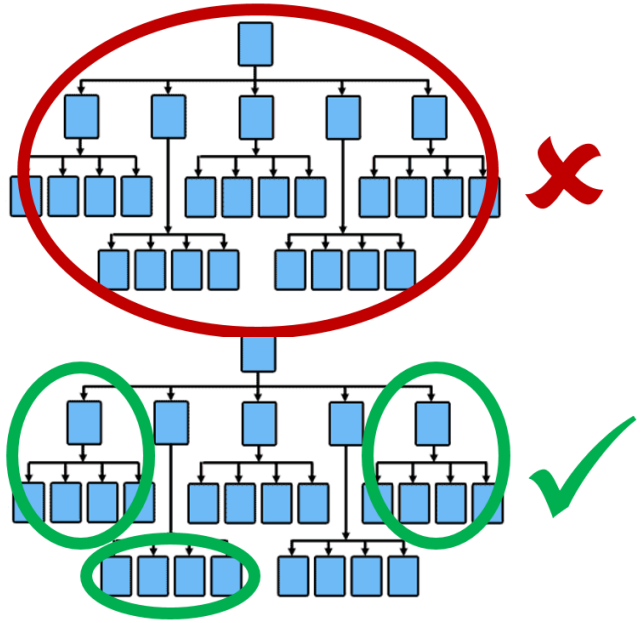
Dealing with state space explosion

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) model the behavior instead of the design

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

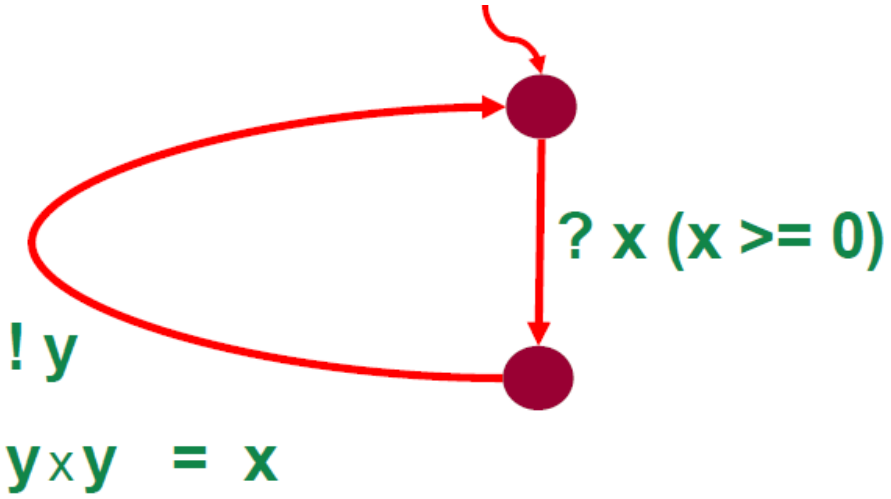
Reference:

- 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:

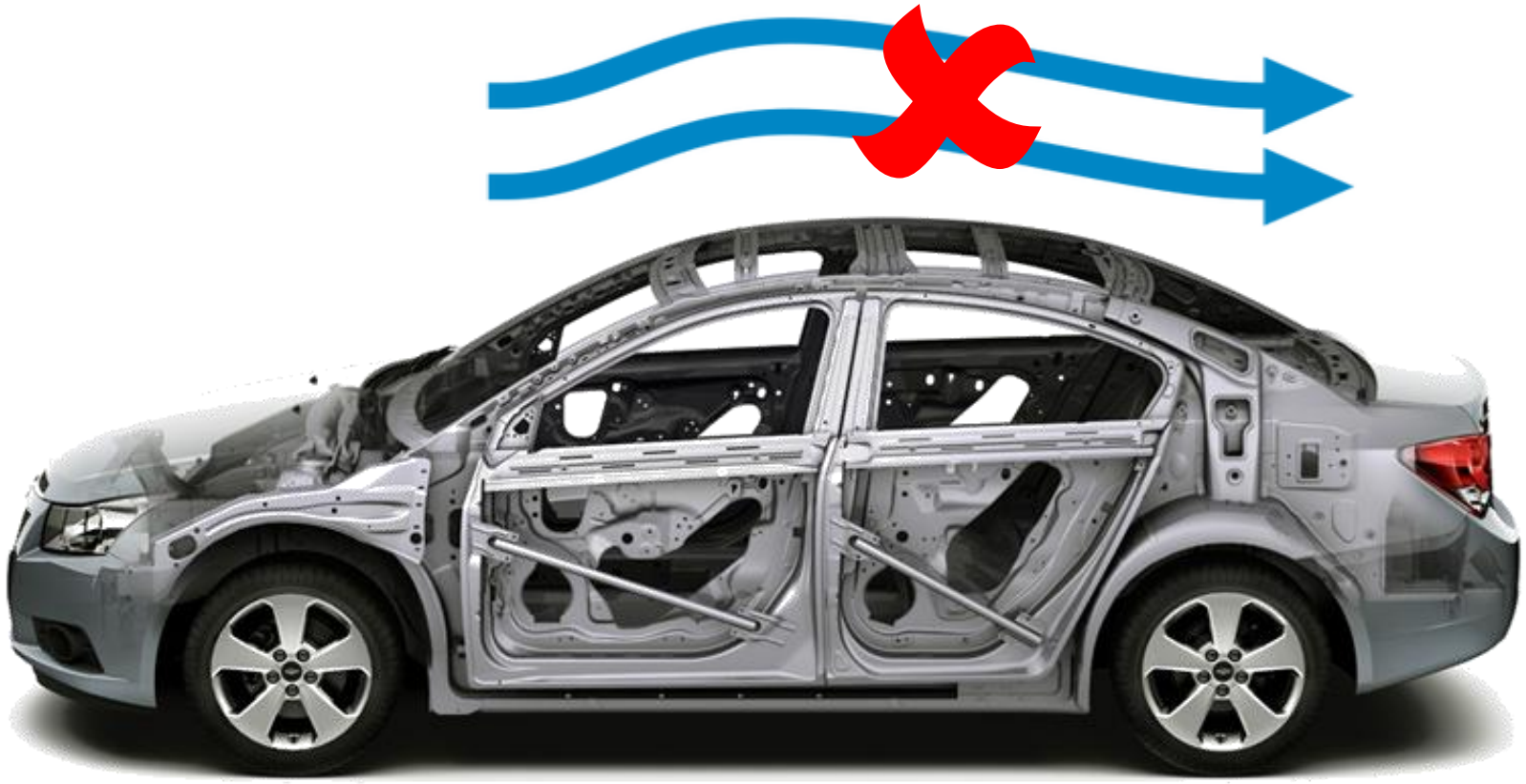
- Jan Tretmans – Radboud University Nijmegen – Model Based Testing

Challenges of MBT – State space explosion

Different models for different purposes

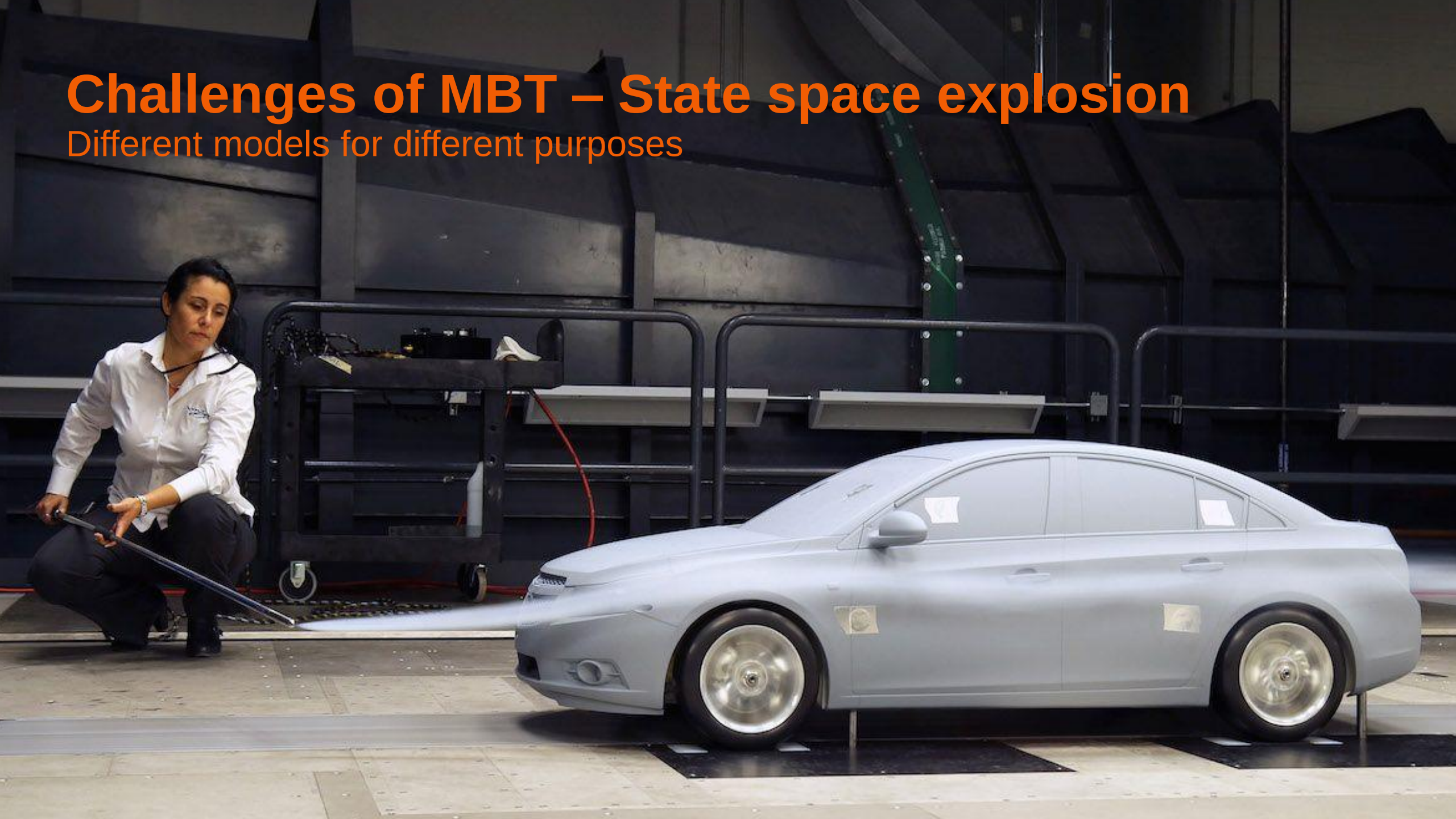
Challenges of MBT – State space explosion

Different models for different purposes

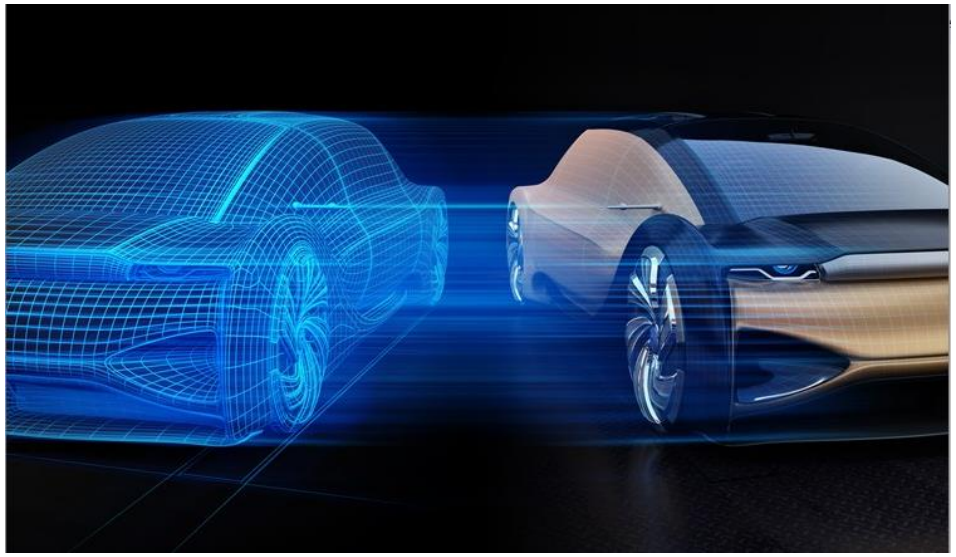


Challenges of MBT – State space explosion

Different models for different purposes



Digital Twin



- Digital twin represents the **physical product** in a **digital world**
  - Design
    - Simulation instead of prototyping (possible to try out much more)
    - Visualize design alternatives
    - Verification of SW inside the twin
    - Less testing on real device
  - Manufacturing
    - Virtual trial runs in digital factory
    - Identify bottlenecks upfront
  - Operation
    - Usage profiles of product and environment
    - Optimize digital twin + environment by using real life data
  - Service
    - Predict and prevent maintenance and downtime

Already extensively
use the product, upfront
In realistic environments

Future?

Future? – Soon, this is not needed anymore

Erlkönig (Camouflaged prototype)

Future? - Iterating in virtual space

- Growing use of digital twins
- Fully virtual target environment
  - Including autonomous driving
  - Weather conditions
  - Pedestrians / bikes
  - Other cars

The Magic Roundabout, Swindon, England

- Thousands of hours of driving, tested within seconds in CI/CD cycle

Future? – Use of machine learning

- Manage state space explosion
- Find the critical/weak SW hot-spots
- Improve virtual environment with data mining
  - Actual info from the field
  - Pedestrians / other cars not behaving as expected

Questions?

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

The Magic Roundabout, Swindon, England