Model-based testing
Overview

- Who am I?
- Why MBT?
- What is MBT?
- MBT theory
- Conclusion
What do you want to hear?
## Comparing levels of test automation

<table>
<thead>
<tr>
<th>Development steps</th>
<th>Manual</th>
<th>Script</th>
<th>MBT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create specification</td>
<td>🧙‍♂️</td>
<td>🧙‍♂️</td>
<td>🧙‍♂️</td>
</tr>
<tr>
<td>Interpret specification</td>
<td>🧙‍♂️</td>
<td>🧙‍♂️</td>
<td>🧙‍♂️</td>
</tr>
<tr>
<td>Create model</td>
<td>-</td>
<td>-</td>
<td>🧙‍♂️</td>
</tr>
<tr>
<td>Create test</td>
<td>🧙‍♂️</td>
<td>🧙‍♂️</td>
<td>✔️</td>
</tr>
<tr>
<td>Predict outcome</td>
<td>🧙‍♂️</td>
<td>🧙‍♂️</td>
<td>✔️</td>
</tr>
<tr>
<td>Script test</td>
<td>-</td>
<td>🧙‍♂️</td>
<td>✔️</td>
</tr>
<tr>
<td>Execute test</td>
<td>🧙‍♂️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Check outcome</td>
<td>🧙‍♂️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

- 🧙‍♂️ manual step
- ✔️ automated step
- use the model as specification

100% automation

More coverage = More certainty
1. Digitize your specification

2. Validate and simulate

3. Automatically generate and execute full coverage test scenarios

4. Get insight into system quality

5. Update and retest with minimal effort
Introducing Axini

- Goal: industrialize model-based testing (MBT) as a highly-rewarding step towards model-based engineering
- Foundation: 25+ years of R&D
- Result-driven and fully funded by commercial MBT
- Proven technology since 2007
Customer base and motivation

✓ Shorten release cycles, increase predictability
✓ Prevent production issues
✓ Reduce TCO

Several high tech companies
- Technical interfaces and protocols
- Ease system integration: single truth for all parties
- Cover timing, parallelism, robustness, bad weather

Top-3 bank, top-3 insurer
- Complex business logic with large data sets
- Cover unique situations, find hard to detect errors
- Simulate changes before implementing them
<table>
<thead>
<tr>
<th>Single</th>
<th>From: Rotterdam Centraal</th>
<th>1st class</th>
<th>Full fare</th>
<th>Valid today</th>
<th>1 ticket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day Return</td>
<td>To: Amsterdam Centraal</td>
<td>2nd class</td>
<td>Discount</td>
<td>Open date</td>
<td>2 tickets</td>
</tr>
<tr>
<td>5 Return ticket</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3 tickets</td>
</tr>
<tr>
<td>Weekend Return</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4 tickets</td>
</tr>
<tr>
<td>Railrunner 4-11 (incl.) years</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To change route: press a white box above.

Select number of tickets

Other tickets | 'Via' station

Nederlands | English

Stop | Clear all
Test case 1

1. Single
2. From
3. To
4. 1st
5. full
6. today
7. 1
Test case 1-5

- Single
- From
- To
- 1st
- full
- today

5 1 2 3 4 5
Test case 1-10

1. Single
2. From
3. To
4. 1st
5. full
6. today
7. No date
Test case 1-30732800

Single

From

Aalten

Zwolle

To

Aalten

Zwolle

1st

full

today

rail runner

Day

No date

Discount

5
x
392
x
392
x
2
x
2
x
5

1 2 3 4 5

= 30732800
Combinatorial explosion
The trouble with testing

Data

Interaction

1. User
   - userName
   - password
   - email
   - phone
   - creditCardInfo
   - shippingInfo
   - registered
   - login

2. SessionManager
   - session
   - username
   - email
   - role
   - login
   - logout

3. Customer
   - customerName
   - address
   - email
   - phone
   - creditCardInfo
   - shippingInfo
   - registered
   - login

4. Product
   - productName
   - description
   - price
   - stock

5. Order
   - orderID
   - dateCreated
   - dateUpdated
   - customerName
   - status
   - shippingInfo
   - orderDetails

6. OrderDetail
   - orderID
   - productID
   - productName
   - stock
   - price
   - shippingInfo

7. Cart
   - cartID
   - orderID
   - productID
   - productName
   - stock
   - price
   - shippingInfo

8. RecommendationSet
   - products
   - name

9. loginButton
   - JButton

10. clearButton
   - JButton

11. usersList
   - List

Flow:
1. ValidateUser
2. displayAuthentication
3. clear
Smart-lock
Testing parallel components
Combinatorial explosion
Theory: compositionality

Spec A

Component A

Spec B

Component B
Compositionality - theory

• When you specify your components precisely
• And you test the components thoroughly in isolation
• Then you do not have to test the integration of the components
Compositionality - theory

- When you specify your components **precisely**
- And you test the components **thoroughly** in isolation
- Then you do not have to test the integration of the components
Modeling in practice

Input: Paper specifications

Step 1: Combine and structure

Step 2: Add missing information
Add bad/sad weather

Missing knowledge

New

Complete digital model
Questions?
Transition Systems

Transition System

- states
- initial state
- labels
- transitions

Notation: ! = response, ? = stimulus
loco, notion of correctness

\[ \text{out}(s) = \{ \lambda \in U_\delta \mid s \xrightarrow{\lambda} \} \]

\[(s \text{ after } \sigma) = \{ s' \mid s \xrightarrow{\sigma} s' \} \]

\[\text{Straces}(s) = \{ \sigma \in L_*^\delta \mid s \xrightarrow{\sigma} s' \} \]

\[\forall \sigma \in \text{Straces}(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]
$$i \text{ioco} s \ =_{\text{def}} \ \forall \sigma \in Straces(s) : \ out(i \ after \ \sigma) \subseteq out(s \ after \ \sigma)$$

$$out(i \ after \ ?dub) = \{!\text{coffee}\} \quad out(s \ after \ ?dub) = \{!\text{coffee}, !\text{tea}\}$$
i \text{ioco} s \overset{\text{def}}{=} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)

\text{out}(i \text{ after } ?\text{dub}) = \{!\text{coffee}, !\text{tea}\} \not\subseteq \text{out}(s \text{ after } ?\text{dub}) = \{!\text{coffee}\}
\[
i \ ioco \ s \ =_{def} \ \forall \sigma \in \ Straces(s) : \ out(i \ after \ \sigma) \subseteq out(s \ after \ \sigma)
\]

\[
\text{out}(i \ after \ ?\text{dub}) = \{!\text{coffee}\} \\
\text{out}(i \ after \ ?\text{kwart}) = \{!\text{tea}\} \\
\text{out}(s \ after \ ?\text{dub}) = \{!\text{coffee}\} \\
\text{out}(s \ after \ ?\text{kwart}) = \emptyset
\]

But \ ?\text{kwart} \not\in \ Straces(s)
Implementation Relation $i \text{ioco} s$

$$i \text{ioco} s \overset{\text{def}}{=} \forall \sigma \in Straces(s) : \text{out} (i \text{ after } \sigma) \subseteq \text{out} (s \text{ after } \sigma)$$

\begin{align*}
\text{out} (i \text{ after } ?\text{kwart}) & = \{ \delta \} \\
\text{out} (s \text{ after } ?\text{kwart}) & = \{ !\text{tea} \}
\end{align*}
\[ i \text{ ioco } s =_{\text{def}} \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \]

\[
\begin{align*}
\text{out}(i \text{ after } ?\text{dub}) &= \{ \delta, !\text{coffee} \} \\
\text{out}(s \text{ after } ?\text{dub}) &= \{ \delta, !\text{coffee} \}
\end{align*}
\]
Implementation Relation $\mathit{ioco}$

\[ \mathit{ioco} \mathit{s} \quad =_{\mathit{def}} \quad \forall \sigma \in \mathit{Straces}(s) : \mathit{out}(\mathit{i after } \sigma) \subseteq \mathit{out}(\mathit{s after } \sigma) \]

\[ \mathit{out}(\mathit{i after } ?\mathit{dub} . ?\mathit{dub}) = \mathit{out}(\mathit{s after } ?\mathit{dub} . ?\mathit{dub}) = \{ !\mathit{tea}, !\mathit{coffee} \} \]

\[ \mathit{out}(\mathit{i after } ?\mathit{dub} . \overline{\delta} . ?\mathit{dub}) = \{ !\mathit{coffee} \} \neq \mathit{out}(\mathit{s after } ?\mathit{dub} . \overline{\delta} . ?\mathit{dub}) = \{ !\mathit{tea}, !\mathit{coffee} \} \]
Our toolset

- Data
- Time
- Functions
- Parallelism
- Non-determinism
- On the fly and off line test-generation
- Test-generation strategies
- Model-checking/validation
- Test-case analysis
Questions?
MBT: Effect on development

![Bar chart comparing MBT and Traditional development efforts](chart.png)

- **MBT**
  - MBT
  - Design
  - Code
  - Test & Integration

- **Traditional**
  - MBT
  - Design
  - Code
  - Test & Integration

**Effort (Traditional=100)**
With MBT you find more bugs

Development  System test  Acceptance test  Production

Found bugs

Bugs not found, but can be found with MBT
Without MBT: long lead time

Development  \[\text{weeks}\]  System test  \[\text{months}\]  Acceptance test  \[\text{months}\]  Production

Acceptance test  \[\text{ACCEPTANCE}\]  valid

Back
With MBT: short cycles, less rework

Development → MBT → Acceptance test → Production

- Development: 20x hours
- MBT: 1x month
- Acceptance test: back
- Production: illustrative

Standard:

<table>
<thead>
<tr>
<th>Development</th>
<th>MBT</th>
<th>Acceptance test</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>120</td>
<td>60</td>
<td>250d</td>
</tr>
</tbody>
</table>

MBT:

<table>
<thead>
<tr>
<th>Development</th>
<th>MBT</th>
<th>Acceptance test</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>60</td>
<td>0</td>
</tr>
</tbody>
</table>

95d
Boehm: cost of errors

Example: Production problems are most expensive to fix.

Example: faults during coding phase are cheap (relatively) to fix.

Example: faults in design are cheapest to fix.
Effect of MBT

• Early fault detection in specification
  – Modeling
  – Inspection
  – Simulation
• Early fault detection in implementation
  – Fast and thorough testing
• Ideal for
  – Agile testing, regression testing
  – Mission critical systems
  – Certification
Questions?