4TEST, AN AGILE MBT METHOD

Presented by István Forgács

4Test-Plus
OVERVIEW

- Traditional test design
- MBT and test design techniques
- Gherkin-based modelling
- How to use 4Test
INTRODUCTION – WHAT MBT SHOULD DO

• Generate reliable test cases very fast
• Prevent design errors
• Support test first method
• Models have to be implementation independent
• Support existing test design techniques
• Should be involved in agile methodology
TRADITIONAL TEST DESIGN
TEST DESIGN TECHNIQUES

- EP+BVA
- state transition testing
- decision table testing
- use case testing
- exploratory testing
- combinatorial testing
- etc.

- in most cases the first two are enough
IN PRACTICE MORE METHODS ARE NEEDED

- Stateless code is rare – STT is needed
- EP, BVA are always needed
- We have to combine EP, BVA, STT
- Not easy – testers with ISTQB exams had problems
- Example: Bus Ticket Vending Machine
Bus Ticket Vending Machine

Initially, the machine offers to display the next available departure time. Selecting this only option the first departure time is displayed, and we have three possibilities:

• Buy ticket
• Go back to the initial screen
• Display the next available departure time

Selecting the last one, the subsequent departure time is displayed and we also select the three possibilities above. However, after the 10\textsuperscript{th} departure time instead of showing the 11\textsuperscript{th}, the machine goes back to the starting state. Selecting the first possibility, the machine asks the number of tickets to buy. We can only select a number of tickets, which are available. After we select the number of tickets, the machine shows the price of the tickets.

Then we have to insert the money. The machine accepts coins and banknotes, i.e. €5, €10, €20, €50 and €100. The machine accepts only those banknotes for which selecting a smaller value banknote would not reach the necessary amount. €5 is always accepted. This maximum banknote is always displayed. When we insert a coin or the acceptable banknote, the remaining tickets price to be inserted is displayed. When the inserted money reaches or exceeds the price of the tickets, they are printed, the change money is given back and the machine goes back to the initial state.
STATE TRANSITION DIAGRAM

Start

Show departure

next departure

first departure

exit

11th departure

Number of tickets

buy tickets

insert number of tickets

insert money: y >= x

Remaining price to be inserted: x

Print tickets, gives change

insert money: y < x;
## EQUIVALENCE PARTITIONS

<table>
<thead>
<tr>
<th>EP</th>
<th>Ticket price EUR</th>
<th>Acceptable banknotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>&lt;=5</td>
<td>€5</td>
</tr>
<tr>
<td>#2</td>
<td>&gt; 5 and &lt;=10</td>
<td>€5, €10</td>
</tr>
<tr>
<td>#3</td>
<td>&gt; 10 and &lt;=20</td>
<td>€5, €10, €20</td>
</tr>
<tr>
<td>#4</td>
<td>&gt; 20 and &lt;=50</td>
<td>€5, €10, €20, €50</td>
</tr>
<tr>
<td>#5</td>
<td>&gt; 50</td>
<td>€5, €10, €20, €50, €100</td>
</tr>
</tbody>
</table>
# BVA TEST CASES

<table>
<thead>
<tr>
<th>Test case</th>
<th>EP</th>
<th>Ticket price EUR</th>
<th>Acceptable banknotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>#1</td>
<td>5</td>
<td>€5</td>
</tr>
<tr>
<td>T2</td>
<td>#2</td>
<td>5.1</td>
<td>€5, €10</td>
</tr>
<tr>
<td>T3</td>
<td>#2</td>
<td>10</td>
<td>€5, €10</td>
</tr>
<tr>
<td>T4</td>
<td>#3</td>
<td>10.1</td>
<td>€5, €10, €20</td>
</tr>
<tr>
<td>T5</td>
<td>#3</td>
<td>20</td>
<td>€5, €10, €20</td>
</tr>
<tr>
<td>T6</td>
<td>#4</td>
<td>20.1</td>
<td>€5, €10, €20, €50</td>
</tr>
<tr>
<td>T7</td>
<td>#4</td>
<td>50</td>
<td>€5, €10, €20, €50</td>
</tr>
<tr>
<td>T8</td>
<td>#5</td>
<td>50.1</td>
<td>€5, €10, €20, €50, €100</td>
</tr>
</tbody>
</table>
## SATISFYING BVAs BY ONE TEST CASE

<table>
<thead>
<tr>
<th>Inserted money</th>
<th>Remaining amount to be inserted before</th>
<th>Remaining amount to be inserted after</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.9</td>
<td>60</td>
<td>50.1</td>
</tr>
<tr>
<td>0.1</td>
<td>50.1</td>
<td>50</td>
</tr>
<tr>
<td>29.9</td>
<td>50</td>
<td>20.1</td>
</tr>
<tr>
<td>0.1</td>
<td>20.1</td>
<td>20</td>
</tr>
<tr>
<td>9.9</td>
<td>20</td>
<td>10.1</td>
</tr>
<tr>
<td>0.1</td>
<td>10.1</td>
<td>10</td>
</tr>
<tr>
<td>4.9</td>
<td>10</td>
<td>5.1</td>
</tr>
<tr>
<td>0.1</td>
<td>5.1</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
TEST CASES

T1: Start:30, Show, Number:1, Remaining:30, *insert money*: 30, Print, Start, Show, Number:1, Remaining:30, Print, Start

T2: **Start**:30, **Show**, **Start**, **Show**, Number:1, Remaining: 30, *insert money*: 30, Print, Start

T3: Start:30, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, **Show**, Number:1, Remaining: 30, *insert money*: 30, Print, Start

T4: Start: 30, Show, Show, Show, Show, Show, Show, Show, Show, Show, Show, **Show**, **Start**, Show, Number:4, Remaining: 120, *insert money*: 120, Print, Start
BVA TEST CASE

T5: Start: 30, Show, Number: 2, Remaining: 60, Max banknote: 100, insert money: 9.9, Remaining: 50.1, Max banknote: 100, insert money: 0.1, Remaining: 50, Max banknote: 50, insert money: 29.9, Remaining: 20.1, Max banknote: 50, insert money: 0.1, Remaining: 20, Max banknote: 20, insert money: 9.9, Remaining: 10.1, Max banknote: 20, insert money: 0.1, Remaining: 10, Max banknote: 10, insert money: 4.9, Remaining: 5.1, Max banknote: 10, insert money: 0.1, Remaining: 5, Max banknote: 5, insert money: 5, Print, Start
MBT AND TEST DESIGN TECHNIQUES
## MBT MODELS – Z. MICSKEI

<table>
<thead>
<tr>
<th>TOOL</th>
<th>INPUT FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4Test</td>
<td>Custom (Gherkin based)</td>
</tr>
<tr>
<td>BPM-Xchange</td>
<td>BPMN, <strong>UML...</strong></td>
</tr>
<tr>
<td>Conformiq Creator</td>
<td><strong>Activity Diagrams</strong>, DSL</td>
</tr>
<tr>
<td>Conformiq Designer</td>
<td><strong>UML State Machines</strong>, QML</td>
</tr>
<tr>
<td>DTM</td>
<td>custom activity model</td>
</tr>
<tr>
<td>fMBT</td>
<td>Custom (AAL)</td>
</tr>
<tr>
<td>GraphWalker</td>
<td><strong>FSM</strong></td>
</tr>
<tr>
<td>JSXM</td>
<td><strong>EFSM</strong> (Stream X-machines)</td>
</tr>
<tr>
<td>JTorX</td>
<td><strong>LTS</strong></td>
</tr>
<tr>
<td>MaTeLo</td>
<td>Markov chains</td>
</tr>
<tr>
<td>MBTsuite</td>
<td><strong>UML or BPMN</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOOL</th>
<th>INPUT FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISTA</td>
<td>PrT net</td>
</tr>
<tr>
<td>Modbat</td>
<td><strong>EFSM</strong> (Scala-based DSL)</td>
</tr>
<tr>
<td>ModelJUnit</td>
<td><strong>EFSM</strong></td>
</tr>
<tr>
<td>MoMuT::UML</td>
<td><strong>UML state machines</strong>, OOAS</td>
</tr>
<tr>
<td>OSMO</td>
<td>model program in Java</td>
</tr>
<tr>
<td>PyModel</td>
<td>Python source</td>
</tr>
<tr>
<td>RT-Tester</td>
<td><strong>UML/SysML, Matlab</strong></td>
</tr>
<tr>
<td>Smartesting CertifyIt</td>
<td><strong>UML + OCL</strong></td>
</tr>
<tr>
<td>Spec Explorer</td>
<td>model programs in C#</td>
</tr>
<tr>
<td>Tcases</td>
<td>Custom</td>
</tr>
<tr>
<td>TEMPPPO</td>
<td>Task flow model</td>
</tr>
<tr>
<td>TestCast</td>
<td><strong>UML State Machines</strong></td>
</tr>
</tbody>
</table>
MBT WITH TEST DESIGN TECHNIQUES

- Most models modelling the system and not the tests
- To include test design technology (EP, BVA) we should
  - Insert sub-models
INSERTING SUBMODELS

Generates different number of tests
Test case explosion
INSERTING TABLES

- Difficult to understand
- Combination of two such cases is difficult
INSERTING CODE

• Models are not understandable for non-IT people
• Model code – graphical and code models may conflict, maintenance is difficult
• To our knowledge: no existing solution
• Test case code – models contain only one part of the specification, less understandable, odd
GHERKIN-BASED MODELLING
PRICE REDUCTION

For online book purchasing, regular customers with cards obtain a 10% price reduction.

Similarly, any customer buying books for at least EUR 50 gets a 10% price reduction.

If somebody has a card and buys books for at least EUR 50, then the price reduction is 15%.

The total book's price appears on the screen.
Categories and choices

• card owner (I): yes (S); no
• book price (I): 49.99; 50 (S)
• price reduction (O): 10%; 15%; no reduction
• total price (O): 45; 44.99; 42.5; 49.99
GENERATED TESTS WITHOUT CONSTRAINTS

For online book purchase, price reduction.
Similarly, any customer price reduction.
If somebody has a card, reduction is 15%.
The total book's price:

T1: card owner(\(I\)) = no, book price(\(I\)) = 49.99
T2: card owner(\(I\)) = yes, book price(\(I\)) = 49.99
T3: card owner(\(I\)) = no, book price(\(I\)) = 50
CONSTRAINTS

Card owner: WHEN card owner IS yes AND book price IS 49.99 THEN price reduction IS 10% AND total price IS 44.99

Expensive: WHEN book price IS 50 THEN price reduction IS 10% AND total price IS 45

Both: WHEN card owner IS yes AND book price IS 50 THEN price reduction IS 15% AND total price IS 42.5

No reduction: WHEN book price IS 49.99 AND card owner IS no THEN price reduction IS no reduction AND total price IS 49.99

© All rights reserved
PRICE REDUCTION MODEL AND TESTS

For online book purchasing, regular customers with cards obtain a 10% price reduction.

Similarly, any customer buying books for at least EUR 50 gets a 10% price reduction.

If somebody has a card and buys books for over EUR 50, then the price reduction is 15%.

The total book's price appears on the screen.

Card owner: card owner =yes, book price =49.99, price reduction =10%, total price =44.99
Expensive: card owner =no, book price =50, price reduction =10%, total price =45
Both: card owner =yes, book price =50, price reduction =15%, total price =42.5
No reduction: book price =49.99, card owner =no, price reduction =no reduction, total price =49.99
GENERATED TEST CASES

• Card owner: card owner(I) = yes, book price(I) = 49.99, price reduction(O) = 10%, total price(O) = 44.99
• Expensive: card owner(I) = no, book price(I) = 50, price reduction(O) = 10%, total price(O) = 45
• Both: card owner(I) = yes, book price(I) = 50, price reduction(O) = 15%, total price(O) = 42.5
• No reduction: book price(I) = 49.99, card owner(I) = no, price reduction(O) = no reduction, total price(O) = 49.99
## MANUAL TEST CASES

<table>
<thead>
<tr>
<th>Test cases</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>passed</td>
<td>3 (75%)</td>
</tr>
<tr>
<td>failed</td>
<td>1 (25%)</td>
</tr>
<tr>
<td>passed+failed</td>
<td>4 (100%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result (passed/failed)</th>
<th>Test case</th>
<th>card owner (I)</th>
<th>book price (I)</th>
<th>price reduction (O)</th>
<th>total price (O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>passed</td>
<td>Card owner</td>
<td>yes</td>
<td>49.99</td>
<td>10%</td>
<td>44.99</td>
</tr>
<tr>
<td>failed</td>
<td>Expensive</td>
<td>no</td>
<td>50</td>
<td>10%</td>
<td>45</td>
</tr>
<tr>
<td>passed</td>
<td>Both</td>
<td>yes</td>
<td>50</td>
<td>15%</td>
<td>42.5</td>
</tr>
<tr>
<td>passed</td>
<td>No reduction</td>
<td>no</td>
<td>49.99</td>
<td>no reduction</td>
<td>49.99</td>
</tr>
</tbody>
</table>
MODEL

• Every user story/feature has a test model
• Categories are preconditions, variables, parameters
• Choices are abstract or given values of the variable
• Constraints are relationships among choices
• Models are structured – the model of feature A may contain a test from feature B
• Complex test cases are generated from understandable models
• Models are developed iteratively
• Every choice has to be covered by at least one test case
• Every constraint has to be covered by at least one test case
• Test selection criterion is unique
FEATURES, CATEGORIES AND CHOICES

- Each feature relates to a model
- Categories can
  - Input (I) and (A)
  - Output (O)
  - Feature (F)
- Choices can be abstract or given (value)
- MyCat(I): MyFirstChoice; MySecondChoice; MyLastChoice; 13
**GERKIN**

- GIVEN describes preconditions, and can be omitted.
- WHEN contains the inputs and obligatory
- THEN contains the output and obligatory
- AND connects two GIVEN/WHEN/THEN expressions
- IS/ARE connects a category and a choice of this category, such as *MyCat IS MyChoice*
- WHEN - THEN - WHEN - THEN - WHEN - THEN sequence is possible:
  - WHEN InsertPIN IS wrong THEN message IS wrong PIN
  - WHEN InsertPIN IS good THEN message IS select transition
(D)

• (D) is the default value of the choices.
• A (D) type choice will be selected for test case generation if all the other choices have been in at least one test case

Example

money transfer price (O): 0; 100(D); 1000
IR

- For IR choice will not be considered as covered during the test generation
- A choice may entirely determine the test - other inputs are irrelevant
- Another test case has to be generated with the same choice for which this choice is relevant

Invalid login: WHEN login name IS invalid AND password IS IR THEN login status IS not logged in
MULTI-LAYER STRUCTURE – (F)

- (F) is a category type where the category name is an existing (lower level) feature
- The choices of this category can be the test case/constraint names

login (F): success; faulty (S)

MyTest: GIVEN login IS success WHEN total price IS 0 THEN paying IS not possible
MULTI-LAYER STRUCTURE – IMPLICIT

\[ \text{login (F): success; faulty (S)} \]

MyTest: GIVEN \text{login IS success} WHEN total price IS 0 THEN paying IS not possible

• Difference – indirect call will not imply login to be involved in other test cases
MULTI-LAYER STRUCTURE – FLEXIBLE

• We can select elements from a sub-model
• We can use them as choices
• Tables reducing the number of constraint if they differ in only in the choices

WHEN original price IS 20 AND reduction is 10 THEN total price IS 19
WHEN original price IS 100 AND reduction is 12 THEN total price IS 88

WHEN original price IS 20 | 100 AND reduction is 10 | 12  THEN total price IS 19 | 88

• Categories can be omitted
SUB-CONSTRAINT

• A constraint, which can be used in other constraints.
• Common sub-models can be used

three items to pay: WHEN number of items IS 3 AND Action IS goto pay

WHEN select food IS pizza AND three items to pay THEN state IS pay

WHEN select food IS fish and chips AND three items to pay THEN state IS pay
GO/GOES

- Makes it possible to model state transition diagrams.

WHEN In IS 1 THEN Out IS yes
WHEN In IS 2 THEN Out IS no
WHEN In IS 3 THEN Out IS N/A

WHEN In GOES 1; 2; 3 THEN
Out GOES yes; no; N/A
ONLY

• **ONLY** + category names describe, which categories remain in a test case.

• For state transition testing it’s essential

• Single ONLY results in that only categories in the constraint are included in the test.

• For sub-constraints the only-s are inherited and united
ONLY EXAMPLE

sub1 (ONLY cat1; cat4): WHEN cat2 IS 1 THEN...
const (ONLY cat7; cat8): WHEN sub1 AND cat6 IS yes THEN

• the generated test case contains:
cat1, cat2, cat4, cat6, cat7, cat8
• and does not contain cat3, cat5
SKIPPED

• SKIPPED + category names result in the listed categories are omitted from the test case.

• Using sub-constraints the skipped categories are inherited

  sub1: WHEN cat2 IS 1 AND cat3 IS SKIPPED
  const: WHEN sub1 AND cat6 IS yes AND cat5 IS SKIPPED

• The generated test case contains : cat1, cat2, cat4, cat6, cat7, cat8 and does not contain cat3, cat5
SOMEOF

• If there are more available choices, which can be selected for a test case we use SOMEOF.
• In this case the generator selects one of the choices following keyword SOMEOF.

WHEN Fruit IS SOMEOF apple; peach; orange; banana THEN Juice IS fine

• The generated test case may contain Fruit = apple or Fruit = peach or Fruit = orange or Fruit = banana.
ANYOF

• In non-deterministic applications the output is not a unique value
• More values may be equally correct
• If the output at the execution is one of them, then the test passed, else failed.

WHEN MyCat IS firstChoice THEN NonDetOutput
ANYOF output1; output2; output 3
HOW TO USE 4TEST
STATE TRANSITION TESTING DEMO

Insert PIN

After inserting a credit card, the machine asks for a PIN code.

If the PIN code is correct, then the account is accessible.

If the PIN code is wrong, then the machine asks again, but after entering the third wrong PIN code the card will be eaten by the cash machine.
STATE TRANSITION DIAGRAM

- **Start**
  - Card inserted
  - Enter wrong PIN less than 3 times
  - Enter wrong PIN third times

- **Wait for PIN**
  - Enter good PIN
  - Access to account

- **Eat Card**
MODEL FOR INSERT PIN

actions (I): insert card; enter wrong PIN; enter good PIN
ATM states (O): wait for PIN; eat card; access to account

wrong PIN twice (ONLY): WHEN actions GOES enter wrong PIN; enter wrong PIN THEN ATM states GO wait for PIN; wait for PIN

success: WHEN actions IS insert card AND wrong PIN twice WHEN actions IS enter good PIN THEN ATM states IS access to account

fail: WHEN actions IS insert card AND wrong PIN twice WHEN actions IS enter wrong PIN THEN ATM states IS eat card

short: WHEN actions IS insert card WHEN actions IS enter good PIN THEN ATM states IS access to account
MODEL AND GENERATED TESTS FOR PIN

After inserting a credit card, the machine asks for a PIN. If the PIN is correct, then the account is accessible. If the PIN is incorrect, the machine asks again, but after entering the third wrong PIN, it will be eaten by the cash machine.

**success:**
- actions: insert card,
- wrong PIN twice [actions: enter wrong PIN, ATM states: wait for PIN, actions: enter wrong PIN, ATM states: wait for PIN, actions: enter good PIN, ATM states: access to account]

**fail:**
- actions: insert card,
- wrong PIN twice [actions: enter wrong PIN, ATM states: wait for PIN, actions: enter wrong PIN, ATM states: wait for PIN, actions: enter wrong PIN, ATM states: eat card]

**short:**
- actions: insert card, actions: enter good PIN, ATM states: access to account
MODEL EDITOR

1. Login(F): successful;
2. card owner (I): yes
3. new book price (I): $149.99
4. second hand book
5. price reduction for bulk
6. total price (O): $149.99; 45; $44.99; $42.5; $142.5

faulty login name
faulty login name(S)
faulty psw
faulty psw(S)
TEST CASES OF INSERT PIN

success: actions (I) = insert card, wrong PIN twice [actions (I) = enter wrong PIN, ATM states (O) = wait for PIN, actions (I) = enter wrong PIN, ATM states (O) = wait for PIN ], actions (I) = enter good PIN, ATM states (O) = access to account

fail: actions (I) = insert card, wrong PIN twice [actions (I) = enter wrong PIN, ATM states (O) = wait for PIN, actions (I) = enter wrong PIN, ATM states (O) = wait for PIN ], actions (I) = enter wrong PIN, ATM states (O) = eat card

short: actions (I) = insert card, actions (I) = enter good PIN, ATM states (O) = access to account
Bus Ticket Vending Machine (BTVM)

Initially, the machine offers to display the next available departure time. Selecting this only option the first departure time is displayed, and we have three possibilities:

• Buy ticket
• Go back to the initial screen
• Display the next available departure time

Selecting the last one, the subsequent departure time is displayed and we also select the three possibilities above. However, after the 10\textsuperscript{th} departure time instead of showing the 11\textsuperscript{th}, the machine goes back to the starting state. Selecting the first possibility, the machine asks the number of tickets to buy. We can only select a number of tickets, which are available. After we select the number of tickets, the machine shows the price of the tickets.

Then we have to insert the money. The machine accepts coins and banknotes, i.e. €5, €10, €20, €50 and €100. The machine accepts only those banknotes for which selecting a smaller value banknote would not reach the necessary amount. €5 is always accepted. This maximum banknote is always displayed. When we insert a coin or the acceptable banknote, the remaining tickets price to be inserted is displayed. When the inserted money reaches or exceeds the price of the tickets, they are printed, the change money is given back and the machine goes back to the initial state.
HOW TO MAKE MODEL FOR BTVM

• Divide and conquer – make models as simple as possible
• Separate EP+BVA from STT
  • Separate “Insert money”
• Let the models be understandable
• We have two sub-features and models:
  • bus ticket
  • insert money
CATEGORIES AND CHOICES FOR BUS TICKETS

price of one ticket (I): 30
bus ticket actions(I): buy; go back; show next departure
number of tickets (I): 1; 2
total price of tickets (O): 30; 60
bus ticket states (O): init screen; next available departure; enter number of tickets; change money back and tickets printed and back to initial screen
maximum banknote(O): 100
10 times departure: WHEN bus ticket actions GO show next departure; show next departure; show next departure; show next departure; show next departure; show next departure; show next departure; show next departure; show next departure; show next departure; THEN bus ticket states GO next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure; next available departure;
Start price 30

first departure

Show departure

buy tickets

Insert number of tickets: 2

Remaining price to be inserted: x
Maximum banknote: y

Insert money: 9.9; 0.1; 29.9; 0.1; 9.9; 0.1; 4.9; 0.1
Remaining money: 50.1; 50; 20.1; 20; 10.1; 10; 5.1; 5

Maximum banknote: 100; 50; 50; 20; 20; 10; 10; 5

Print tickets, gives change

Insert money: 5
EP TEST

EP test: GIVEN 10 times departure WHEN bus ticket actions IS buy THEN bus ticket states IS enter number of tickets WHEN number of tickets IS 2 THEN total price of tickets IS 60 AND maximum banknote IS 100 WHEN insert money IS EUR 60 THEN bus ticket states IS change money back and tickets printed and back to initial screen
# INSERT MONEY — CATEGORIES AND CHOICES

<table>
<thead>
<tr>
<th>insert money (I):</th>
<th>0.1; 4.9; 5; 9.9; 29.9; 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>remaining money (O):</td>
<td>50.1; 50; 20.1; 20; 10.1; 10; 5.1; 5; 0</td>
</tr>
<tr>
<td>maximum banknote (O):</td>
<td>5; 10; 20; 50; 100</td>
</tr>
</tbody>
</table>

- Non-ordered lists
CONTRAINTS

insert remaining EUR 5: WHEN insert money IS 5 THEN remaining money IS 0

EUR 60: WHEN insert money GOES 9.9; 0.1; 29.9; 0.1; 9.9; 0.1; 4.9; 0.1 THEN remaining money GOES 50.1; 50; 20.1; 20; 10.1; 10; 5.1; 5 AND maximum banknote GOES 100; 50; 50; 20; 20; 10; 10; 5 WHEN insert remaining EUR 5

EUR 30: WHEN insert money IS 30 THEN remaining money IS 0
EP TEST CASE

EP test: 10 times departure [bus ticket actions (l) = show next departure, bus ticket actions (l) = show next departure, bus ticket actions (l) = show next departure, bus ticket actions (l) = show next departure, bus ticket actions (l) = show next departure, bus ticket actions (l) = show next departure, bus ticket actions (l) = show next departure], price of one ticket (l) = 30, bus ticket actions (l) = buy, bus ticket states (O) = enter number of tickets, number of tickets (l) = 2, total price of tickets (O) = 60, maximum banknote (O) = 100, insert money (F) = EUR 60 [insert money (l) = 9.9, remaining money (O) = 50.1, maximum banknote (O) = 100, insert money (l) = 0.1, remaining money (O) = 50, maximum banknote (O) = 50, insert money (l) = 29.9, remaining money (O) = 20.1, maximum banknote (O) = 50, insert money (l) = 0.1, remaining money (O) = 20, maximum banknote (O) = 20, insert money (l) = 9.9, remaining money (O) = 10.1, maximum banknote (O) = 20, insert money (l) = 0.1, remaining money (O) = 10, maximum banknote (O) = 10, insert money (l) = 4.9, remaining money (O) = 5.1, maximum banknote (O) = 10, insert money (l) = 0.1, remaining money (O) = 5, maximum banknote (O) = 5, insert remaining EUR 5 [insert money (l) = 5, remaining money (O) = 0], bus ticket states (O) = change money back and tickets printed and back to initial screen.
### GENERATED MANUAL TEST CASE FOR EP+BVA

<table>
<thead>
<tr>
<th>Test case</th>
<th>EP test</th>
</tr>
</thead>
<tbody>
<tr>
<td>null (null)</td>
<td>10 times departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>show next departure</td>
</tr>
<tr>
<td>price of one ticket (I)</td>
<td>30</td>
</tr>
<tr>
<td>bus ticket actions (I)</td>
<td>buy</td>
</tr>
<tr>
<td>bus ticket states (O)</td>
<td>enter number of tickets</td>
</tr>
<tr>
<td>number of tickets (I)</td>
<td>2</td>
</tr>
<tr>
<td>total price of tickets (O)</td>
<td>60</td>
</tr>
<tr>
<td>maximum banknote (O)</td>
<td>100</td>
</tr>
</tbody>
</table>

| insert money (F) | EUR 60 |
| insert money (I) | 9.9 |
| remaining money (O) | 50.1 |
| maximum banknote (O) | 100 |
| insert money (I) | 0.1 |
| remaining money (O) | 50 |
| maximum banknote (O) | 50 |
| insert money (I) | 29.9 |
| remaining money (O) | 20.1 |
| maximum banknote (O) | 50 |
| insert money (I) | 0.1 |
| remaining money (O) | 20 |
| maximum banknote (O) | 20 |
| insert money (I) | 9.9 |
| remaining money (O) | 10.1 |
| maximum banknote (O) | 20 |
| insert money (I) | 0.1 |
| remaining money (O) | 10 |
| maximum banknote (O) | 10 |
| insert money (I) | 4.9 |
| remaining money (O) | 5.1 |
| maximum banknote (O) | 10 |
| insert money (I) | 0.1 |
| remaining money (O) | 5 |
| maximum banknote (O) | 5 |
| null (null) | insert remaining EUR 5 |
| insert money (I) | 5 |
| remaining money (O) | 0 |
| bus ticket states (O) | change money back and tickets printed and back to initial screen |
BENEFITS OF GHERKIN-BASED MODELS

• Real test model – testers can use it
• No test selection criterion has to be set
• Fewer test cases
• Tests are understandable for everybody
• Learning curve is manageable – about 20 hours
• Flexible sub-modelling
• Smooth integration of EP, BVA and STT
THANK YOU!

QUESTIONS?

MORE INFORMATION:
4test.io
forgacs@4test.hu