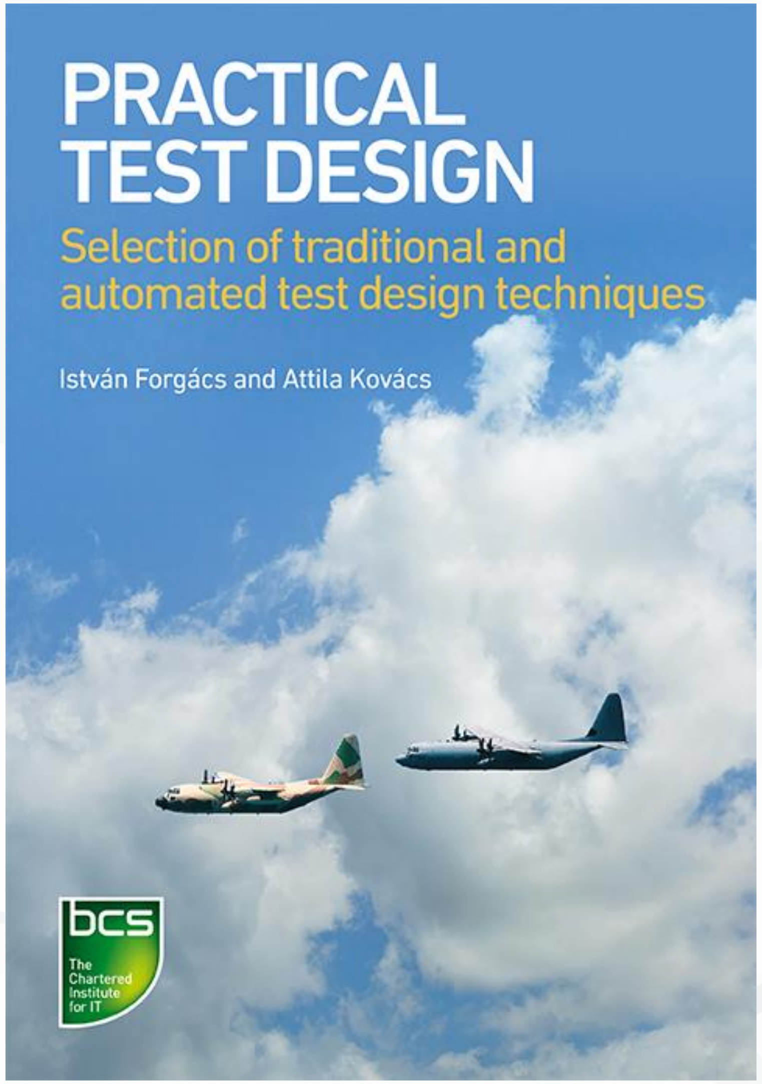
Harmony – requirement management and QA tool

István Forgács PhD
4TestDev
4test.io

Introduction

- Personal introduction
- About 4TestDev

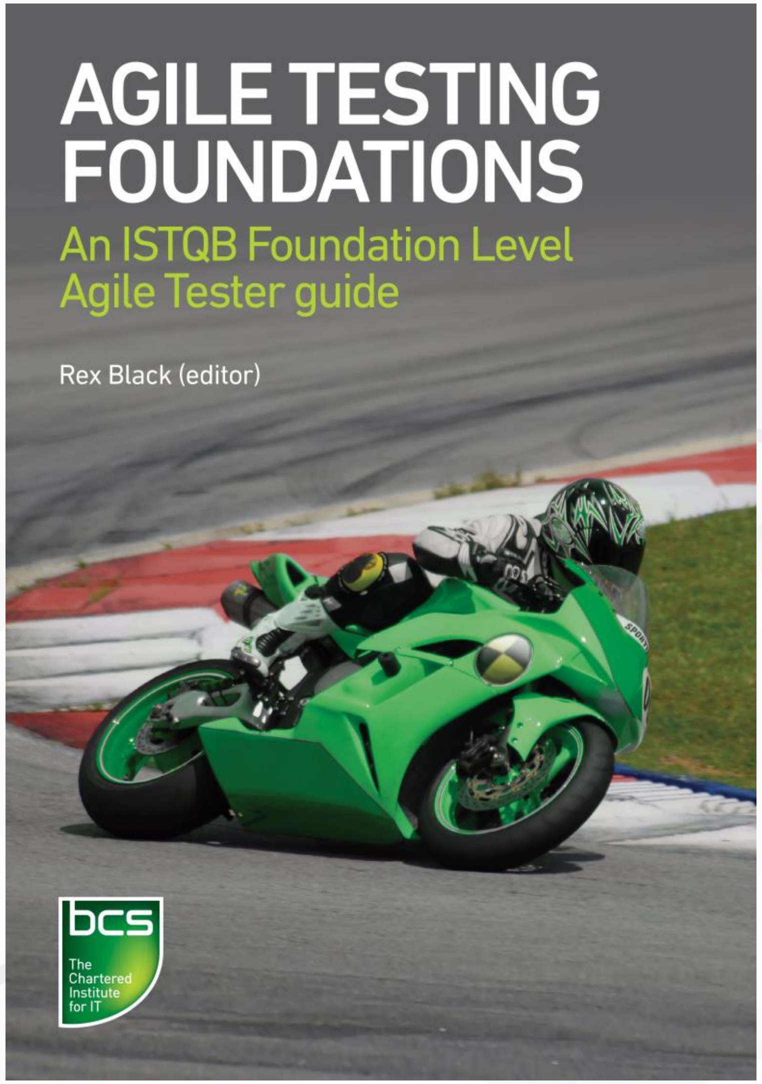


PRACTICAL TEST DESIGN

Selection of traditional and automated test design techniques

István Forgács and Attila Kovács

bcs The Chartered Institute for IT



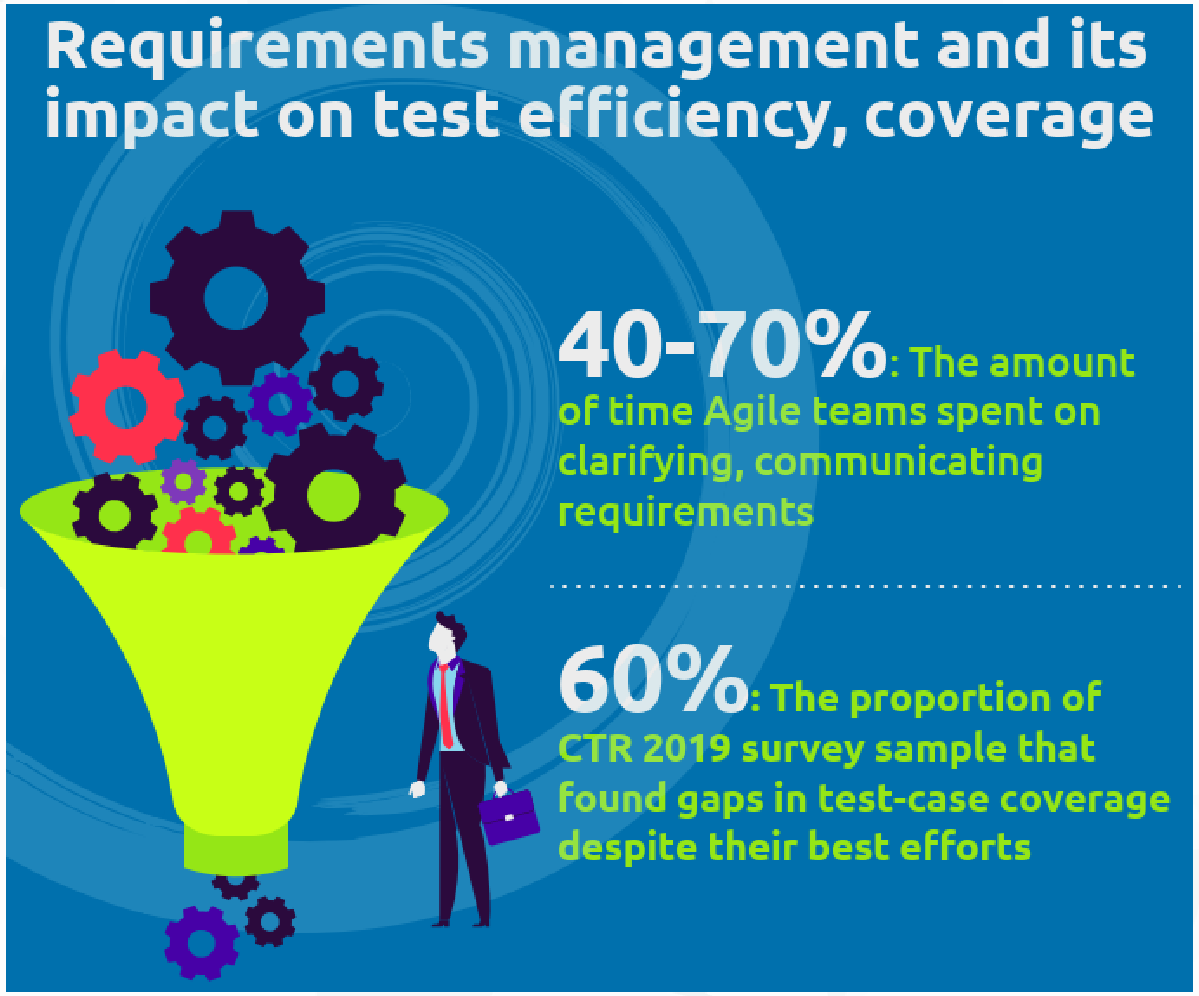
AGILE TESTING FOUNDATIONS

An ISTQB Foundation Level Agile Tester guide

Rex Black (editor)

bcs The Chartered Institute for IT

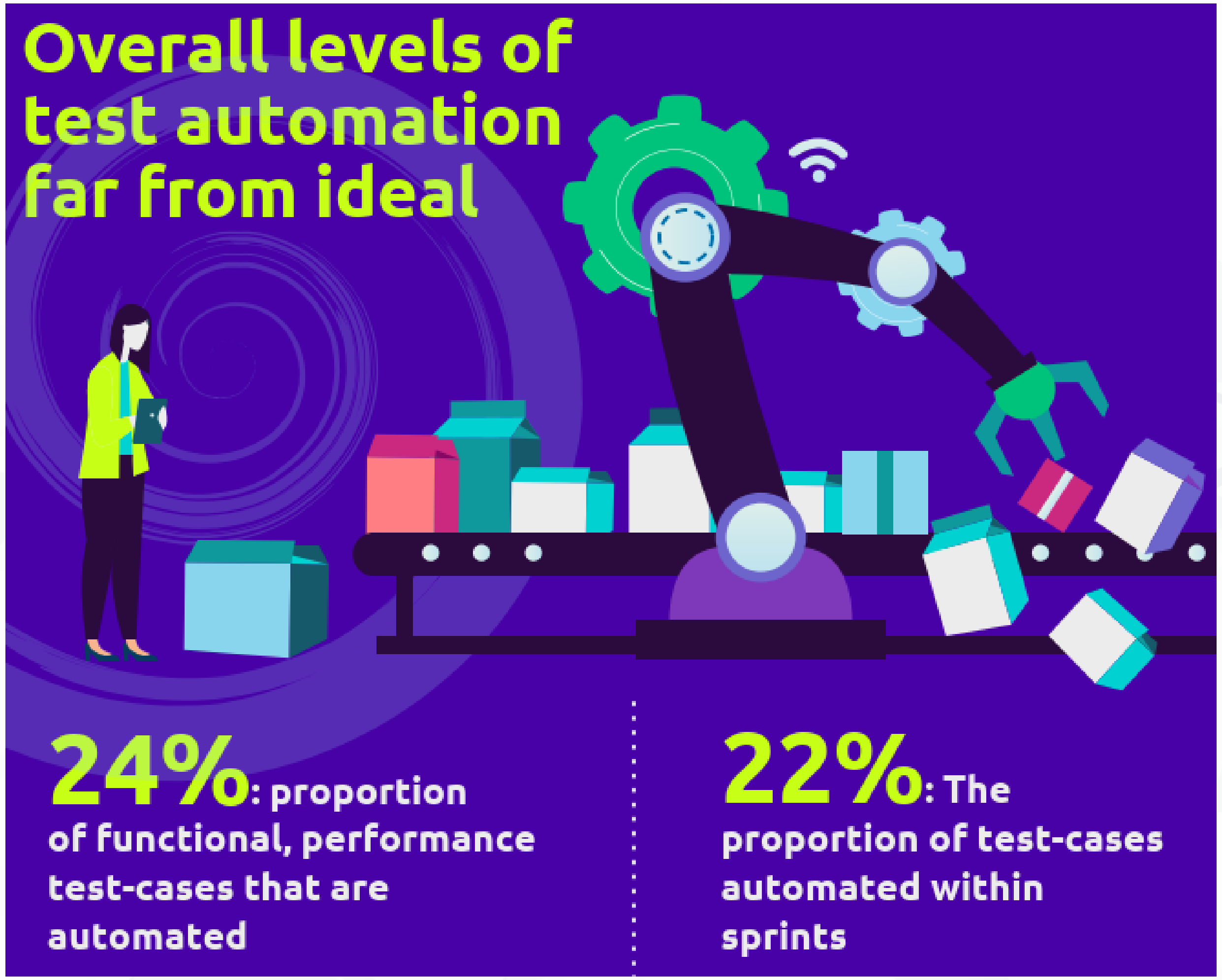
Capgemini survey



Requirements management and its impact on test efficiency, coverage

40-70%: The amount of time Agile teams spent on clarifying, communicating requirements

60%: The proportion of CTR 2019 survey sample that found gaps in test-case coverage despite their best efforts

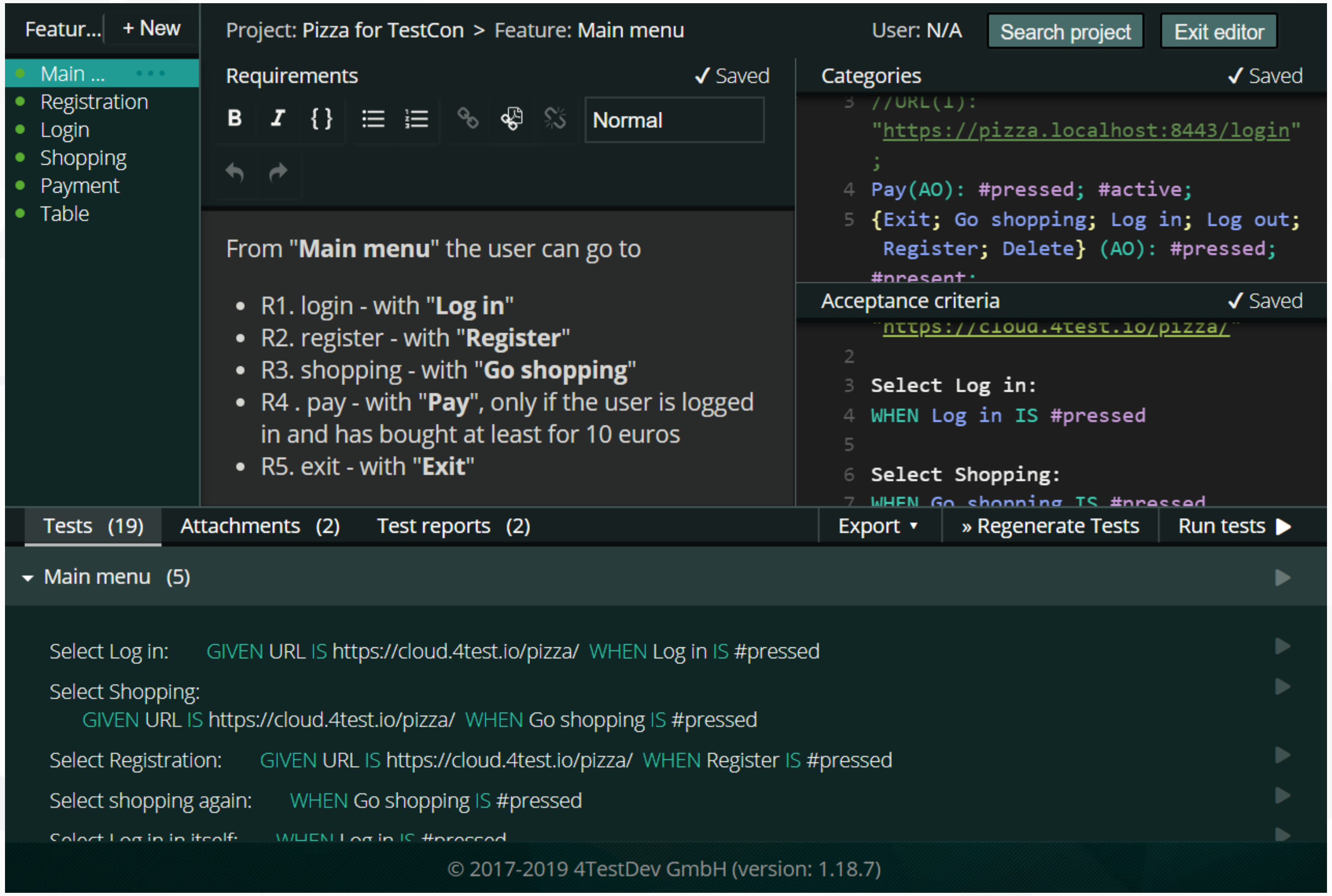


Overall levels of test automation far from ideal

24%: proportion of functional, performance test-cases that are automated

22%: The proportion of test-cases automated within sprints

Part I. Introduction to Harmony



From "**Main menu**" the user can go to

- R1. login - with "**Log in**"
- R2. register - with "**Register**"
- R3. shopping - with "**Go shopping**"
- R4 . pay - with "**Pay**", only if the user is logged in and has bought at least for 10 euros
- R5. exit - with "**Exit**"

```
//URL(1):
"https://pizza.localhost:8443/login"
;
Pay(AO): #pressed; #active;
{Exit; Go shopping; Log in; Log out;
 Register; Delete} (AO): #pressed;
#present;
```

```
"https://cloud.4test.io/pizza/"

Select Log in:
WHEN Log in IS #pressed

Select Shopping:
WHEN Go shopping IS #pressed
```

Select Log in: GIVEN URL IS https://cloud.4test.io/pizza/ WHEN Log in IS #pressed

Select Shopping:
GIVEN URL IS https://cloud.4test.io/pizza/ WHEN Go shopping IS #pressed

Select Registration: GIVEN URL IS https://cloud.4test.io/pizza/ WHEN Register IS #pressed

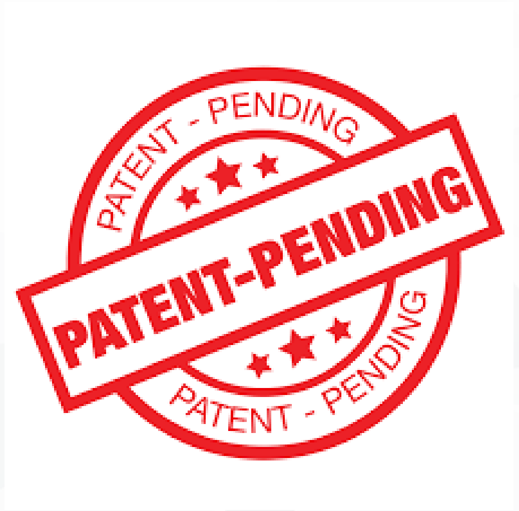
Select shopping again: WHEN Go shopping IS #pressed

Select Log in in itself: WHEN Log in IS #pressed

© 2017-2019 4TestDev GmbH (version: 1.18.7)

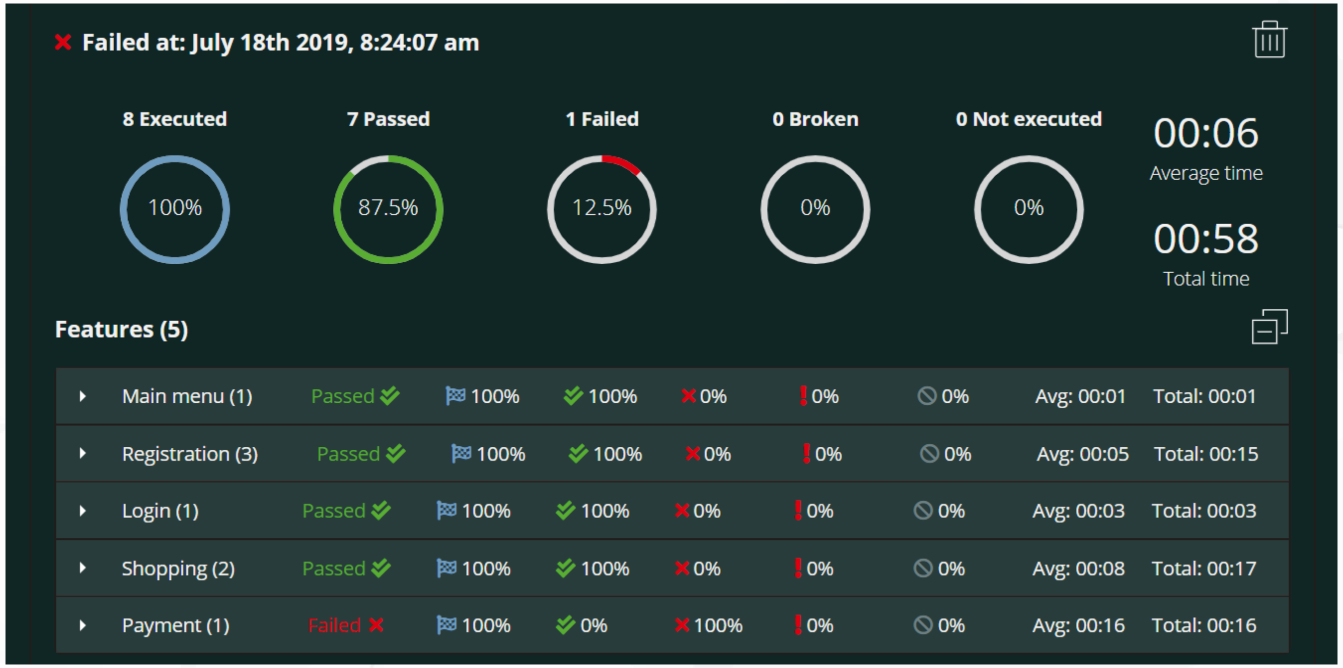
What is Harmony?

- Test design automation tool
- Codeless test automation tool
- DevOps tool
- Requirement management tool
- QA tool



How Harmony is different?

- Test first test automation tool – no application should be ready and executable
- Locators are generated instead of scanning or recording
- Uses a domain specific language, which supports most of the test design techniques
- Requirements are related to acceptance criteria instead of test cases



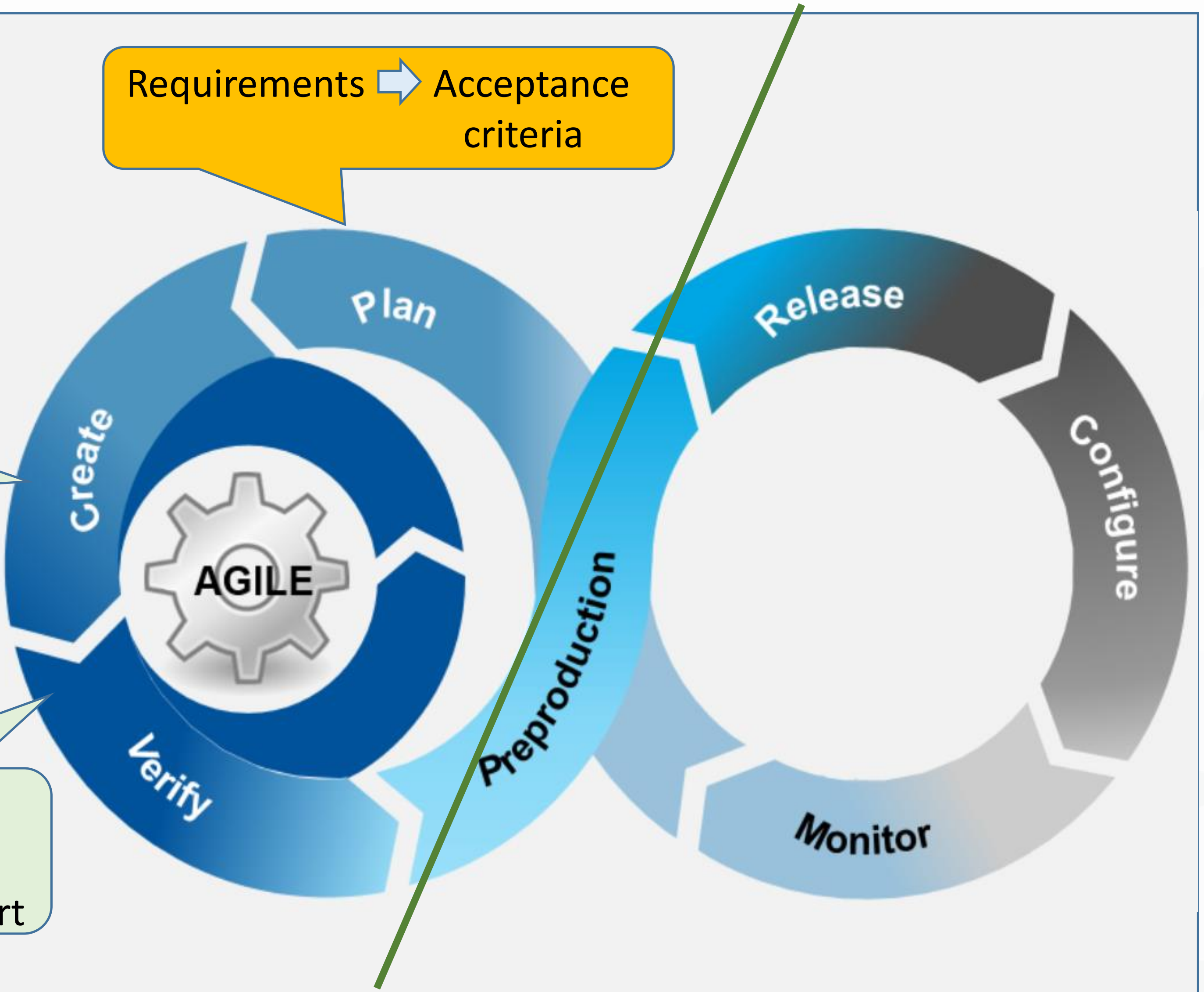
✗ Failed at: July 18th 2019, 8:24:07 am

8 Executed	7 Passed	1 Failed	0 Broken	0 Not executed	
100%	87.5%	12.5%	0%	0%	00:06 Average time
					00:58 Total time

Features (5)

Main menu (1)	Passed ✓	🏁 100%	✓ 100%	✗ 0%	! 0%	⊘ 0%	Avg: 00:01	Total: 00:01
Registration (3)	Passed ✓	🏁 100%	✓ 100%	✗ 0%	! 0%	⊘ 0%	Avg: 00:05	Total: 00:15
Login (1)	Passed ✓	🏁 100%	✓ 100%	✗ 0%	! 0%	⊘ 0%	Avg: 00:03	Total: 00:03
Shopping (2)	Passed ✓	🏁 100%	✓ 100%	✗ 0%	! 0%	⊘ 0%	Avg: 00:08	Total: 00:17
Payment (1)	Failed ✗	🏁 100%	✓ 0%	✗ 100%	! 0%	⊘ 0%	Avg: 00:16	Total: 00:16

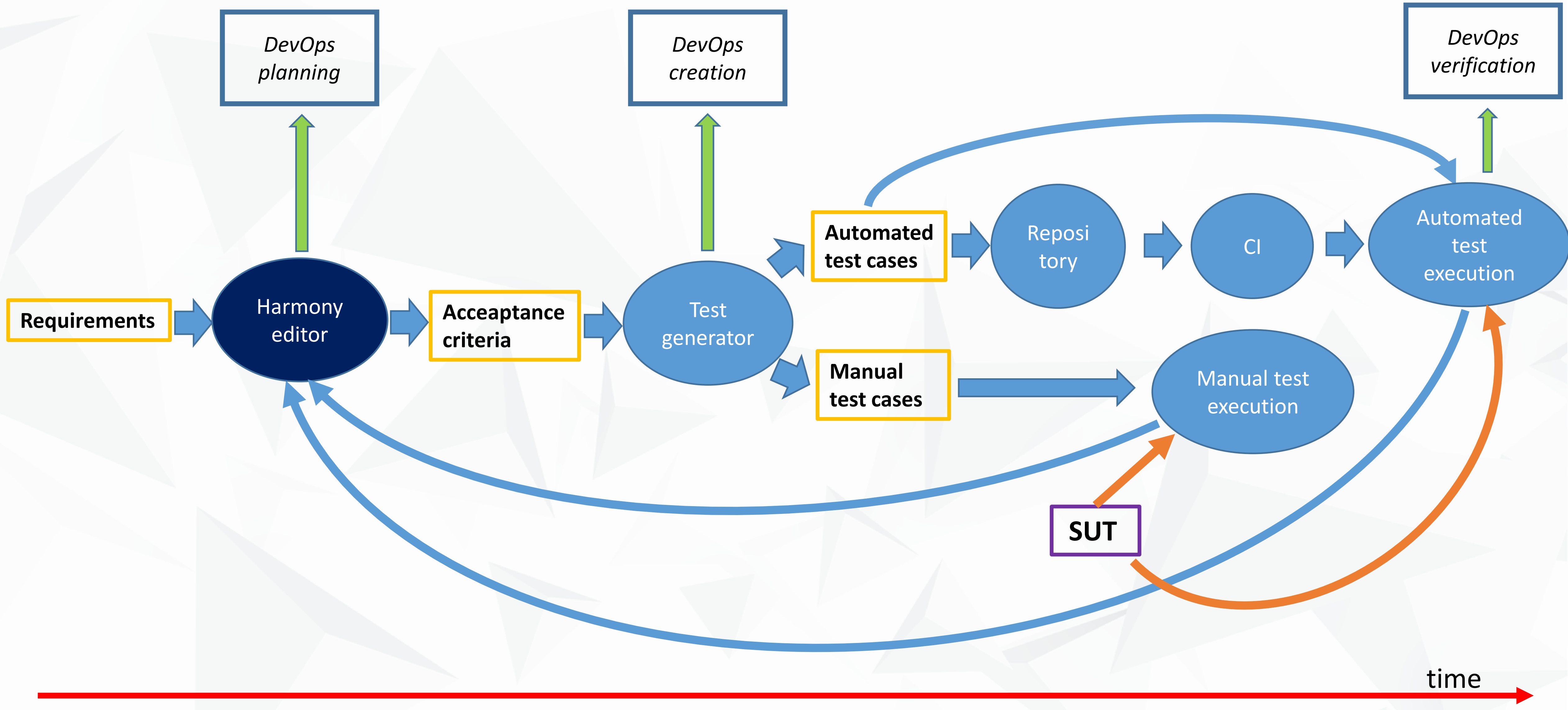
Harmony DevOps solution



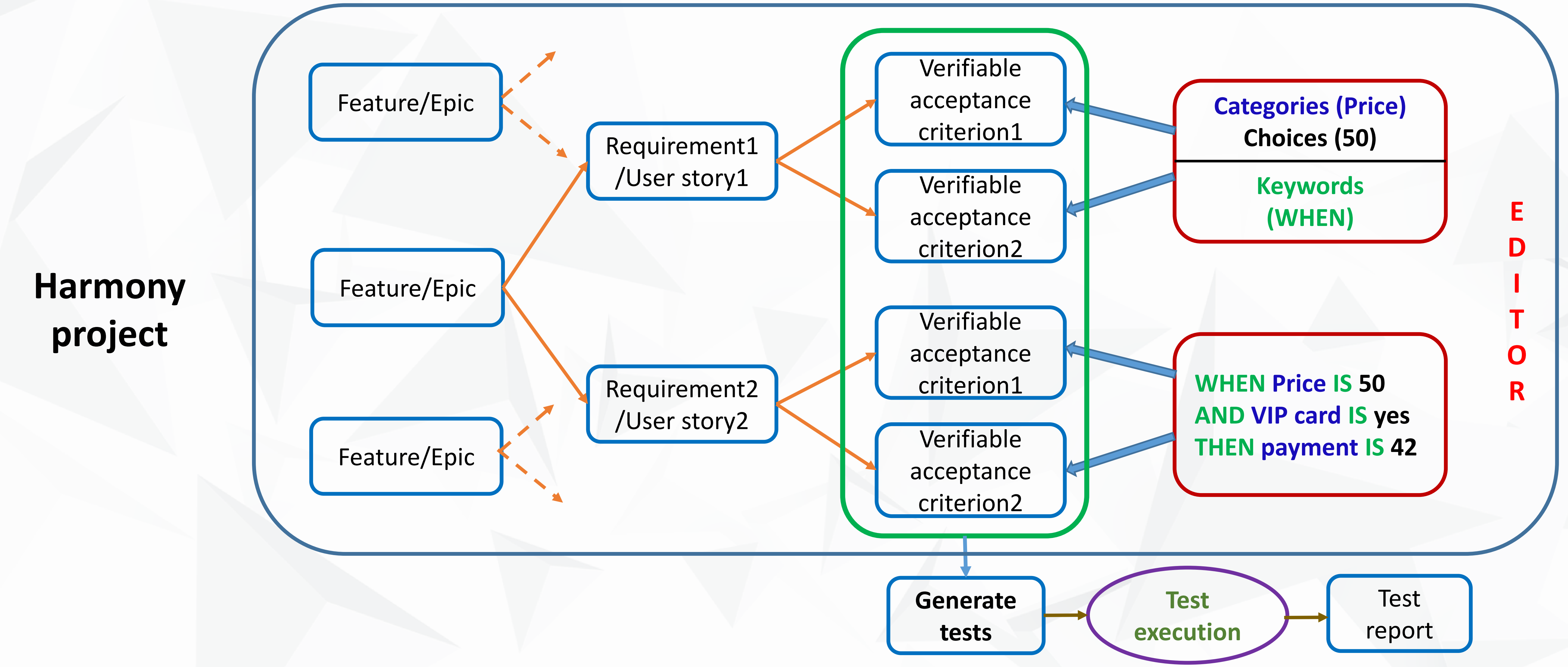
Acceptance criteria
⇩
Automated test cases

Automated test cases
⇩
Test execution and report

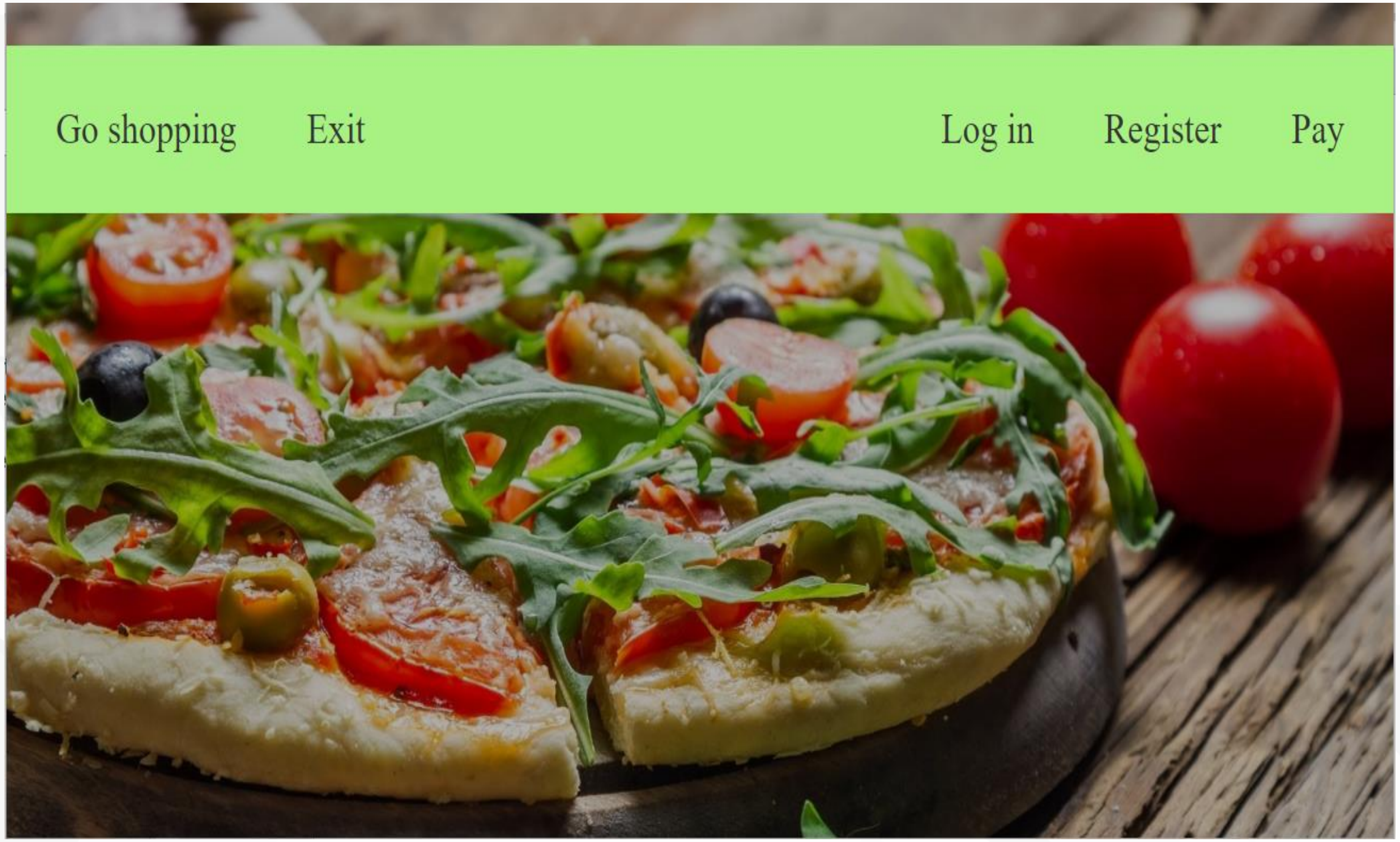
Codeless testing process



Harmony editor



Pizza application – Demo I.



Requirements

Requirements are simple numbered text
R1.
R2.
…
Rn.

R1. The user shall fill 'Login name', otherwise an error message: 'Login name is required' appears.

R2. The user shall fill 'Password', otherwise an error message: 'Password is required' appears.

R3. If either of them is wrong, then an error message: 'Incorrect name or password' appears.

R4. If the login is successful, then a message appears 'Login successful!'

R5. After successful login, pressing a button "Next" enables going back to menu Login has been called.

R6. "Exit" goes back to the main menu without logging in.

Let’s validate requirements – categories and choices

Based on the requirements, categories and choices are planned:

R1. The user shall fill 'Login name', otherwise an error message: 'Login name is required' appears.

R4. If the login is successful, then a message appears 'Login successful!'

- Login name(I): Smith; Wigner
- Password(I): 2a4b6c(D); wrong-PSW
- Msg(O): Login name is required; Password is required; Incorrect name or password; Login successful!
- Next(A): #pressed

Let’s validate requirements – acceptance criteria

Acceptance criteria for a given requirement are statements by which the implemented requirement can be validated. ACs are designed based on the requirements, the categories with choices and the selected test design techniques

- ACs should be good enough to validate the related requirement (complete);
- ACs should be correct ;
- ACs should be testable;
- ACs should be clear and concise;
- ACs must be understandable for everyone.

Acceptance criteria (AC)

From requirements **concrete** acceptance criteria are created.
An AC contains a name and test steps: keyword – category – keyword - choice

R4. If the login is successful, then a message appears 'Login successful!'

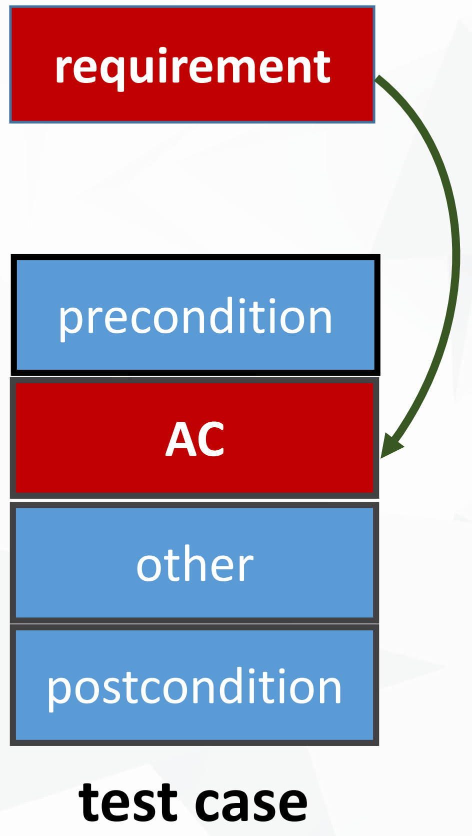
```
Successful:
WHEN Login name IS Smith
AND Password IS 2a4b6c
WHEN Next IS #pressed
THEN Msg IS Login successful!
WHEN Next IS #pressed
```

Acceptance criteria vs test cases

A test case contains the following parts:

- precondition – we should go to feature along a program path
- **acceptance criterion – relate to some requirement(s)**
    - more acceptance criteria, i.e. test cases may relate to one requirement
- other part – input and output not related to the AC
- postcondition – closing the test to go to an appropriate state

- a test case may contain several test steps making it difficult to understand
- we connect the AC part of the test case to the requirement
- this is a better mapping as each part of an AC is relevant w.r.t. the requirement



Make our test runable

We need a precondition to go to Login and a postcondition to logout

```
PRECONDITION Main menu IS Select Log in


Successful:
WHEN Login name IS Smith
AND Password IS 2a4b6c
WHEN Next IS #pressed
THEN Msg IS Login successful!
WHEN Next IS #pressed
         WHEN Main menu IS Exit
```

Generated test case

Successful:

GIVEN Main menu IS Select Log in {GIVEN Platform IS Chrome GIVEN URL IS https://cloud.4test.io/pizza/ GIVEN Log in IS #pressed }

WHEN Login name IS Smith

WHEN Password IS 2a4b6c

WHEN Next IS #pressed THEN Msg IS Login successful!

WHEN Next IS #pressed

WHEN Main menu IS Exit {WHEN Exit IS #pressed }

Test selection criterion

We have 3 input categories: Name(I), Address(I), Payment(I)

WHEN Name IS Smith AND Payment IS 5000

The test generated test case will be:

WHEN Name IS Smith
WHEN Address IS 12. Millenium St
WHEN Payment IS 5000

One test case is generated for each AC and at least one for each input choice.

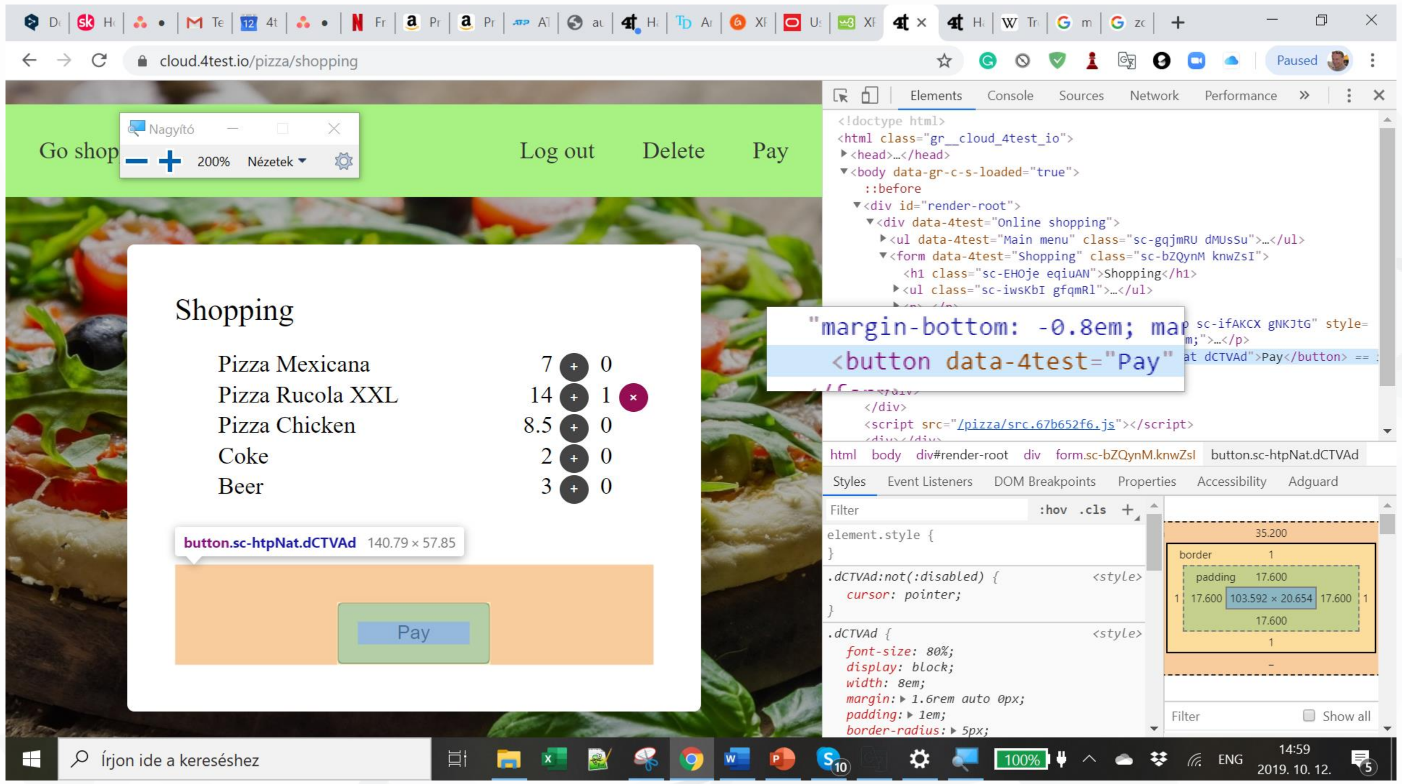
Executable test code generation

Code made by OpenKeyWord:

```
EN.SetValue("URL", "https://cloud.4test.io/pizza/");
EN.SelectWindow("Main menu");
EN.ClickOn("Log in");
EN.SelectWindow("Login");
EN.SetValue("Login name", "Smith");
EN.SetValue("Password", "2a4b6c");
EN.ClickOn("Next");
EN.VerifyValue("Msg", "Login successful!");
EN.ClickOn("Next");
EN.SelectWindow("Main menu");
EN.ClickOn("Exit");
```

Locators are generated

- No capture or scan of the screens
- Initial rule set
- Adapted rule set
- Locators are generated from it
- Lots of potential path expressions are evaluated – one match is enough
- Categories can be extended by XPath path expressions
- Here we use a special attribute: data-4test



Shopping

Pizza Mexicana	7	0
Pizza Rucola XXL	14	1
Pizza Chicken	8.5	0
Coke	2	0
Beer	3	0

button.sc-htpNat.dCTVAd 140.79 × 57.85

Pay

```
<!doctype html>
<html class="gr__cloud_4test_io">
▶<head>…</head>
▼<body data-gr-c-s-loaded="true">
    ::before
  ▼<div id="render-root">
    ▼<div data-4test="Online shopping">
      ▶<ul data-4test="Main menu" class="sc-gqjmRU dMUsSu">…</ul>
      ▼<form data-4test="Shopping" class="sc-bZQynM knwZsI">
          <h1 class="sc-EHOje eqiuAN">Shopping</h1>
        ▶<ul class="sc-iwsKbI gfqmRl">…</ul>
 "margin-bottom: -0.8em; mar  sc-ifAKCX gNKJtG" style=
                              m;">…</p>
  <button data-4test="Pay"    at dCTVAd">Pay</button> ==
    </div>
    <script src="/pizza/src.67b652f6.js"></script>
```

html body div#render-root div form.sc-bZQynM.knwZsI button.sc-htpNat.dCTVAd

```
element.style {
}
.dCTVAd:not(:disabled) {                <style>
  cursor: pointer;
}
.dCTVAd {                               <style>
  font-size: 80%;
  display: block;
  width: 8em;
  margin: ▶ 1.6rem auto 0px;
  padding: ▶ 1em;
  border-radius: ▶ 5px;
```

Test execution

When the acceptance criteria are ready a built-in runner can execute the generated test code using Selenium

- One test case
- Test cases for a feature
- All the test cases

CI

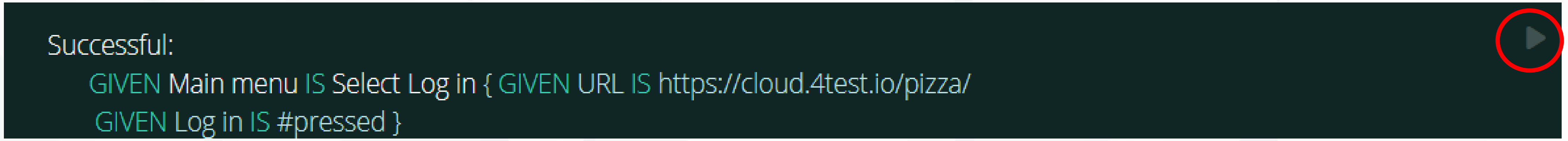
- Test code is generated
- Harmony can export the test code to all CI tools via git
- Test case are executed in a scheduled way

Demo II. – how to use Harmony

First exercise

Execute test case “success” in feature Login

1. Open https://cloud.4test.io
2. Login with your google account
3. Click on icon “add sample projects” on the right hand side
4. Click on “Pizza for workshop”
5. Execute the test case ‘Successful’:



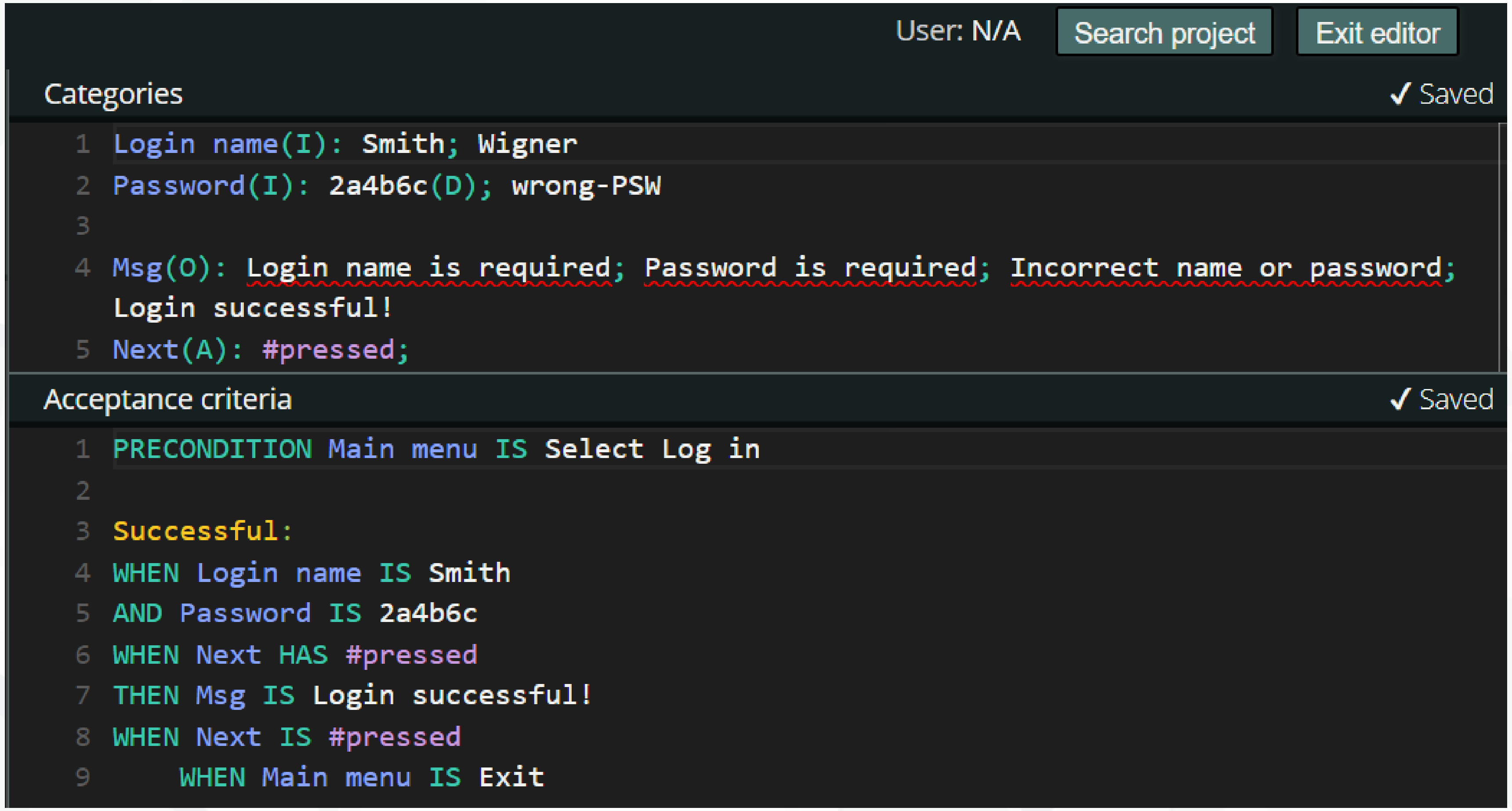
```
Successful:
    GIVEN Main menu IS Select Log in { GIVEN URL IS https://cloud.4test.io/pizza/
     GIVEN Log in IS #pressed }
```

6. Download Runner – the directory containing it cannot contain accent mark (á, ó)
7. Execute Runner – JRE java 8 is required
8. Run the tests again (repeat Step 5)

See Login for the remaining part of this exercise

The exercises are in the related features below the requirements

Part II. Gherkin++



Categories ✓ Saved

```
Login name(I): Smith; Wigner
Password(I): 2a4b6c(D); wrong-PSW

Msg(O): Login name is required; Password is required; Incorrect name or password;
Login successful!
Next(A): #pressed;
```

Acceptance criteria ✓ Saved

```
PRECONDITION Main menu IS Select Log in

Successful:
WHEN Login name IS Smith
AND Password IS 2a4b6c
WHEN Next HAS #pressed
THEN Msg IS Login successful!
WHEN Next IS #pressed
    WHEN Main menu IS Exit
```

Categories (I), (A), (O), (IO), (AO), (F)

Categories are the names of the parameters, states or messages

- yyy(A): – A: action such as press a button
- kkk(O): – AO: output
- vvv(IO):
- www(AO): – IO, AO can be used for both inputs and outputs

Example.

Name(I): Smith; Roth

Login(AO): #pressed; #present

Total price(O): 10; 20

Choices (S), (D)

Choices are values for the categories used in the acceptance criteria and the generated test cases

- (D): default – output values are also generated
- (S): single – generated only once

Example.

Book price(I): 50(S); 49.99; 1

VIP(I): yes; no(D)

Message(O): Successful login(D); Wrong login name or password

Second exercise

- **Make categories with choices.**
- Think about the necessary test cases and create the output choices as well
- Use (D) for generating output values
- Study the generated test cases.

Feature Buy book

R1. For online new book purchasing, regular customers with cards obtain a 10% price reduction.

R2. Similarly, any customer buying new books for at least EUR 50 gets a 10% price reduction.

R3. If somebody has a card and buys new books for at least EUR 50, then the price reduction is 15%.

R4. The price reduction holds only for new books even if the customer buys new and second-hand books together).

R5. The total book's price appears on the screen.

Gherkin

- a domain specific language
- helps to describe business behavior
- acts as documentation and skeleton of your automated tests.

```
Feature: Login
  Scenario: Happy path
    When I insert my account
    And I insert my password
    Then I’m logged in

  Scenario: Wrong password
    When I insert my account
    And I insert a wrong password
    Then a message “wrong login name or password” appears
```

Gherkin and Gherkin++

In Gherkin the test code, i.e. “step definition” is **written** based on the Given-When-Then steps:

```
public class MyStepdefs{
    @When(“I insert my account”)
    public void…
```

Gherkin++ is an extension of Gherkin, which makes it possible:

- to automated the test code generation,
- to involve most of the test design techniques

Basic Gherkin++ keywords

- PRECONDITION is used for setting some initial states, which is necessary for the acceptance criteria followed it.
  - Ends with another PRECONDITION or PRECONDITION END
- GIVEN describes preconditions, and can be omitted.
- INITIALLY describes initial state – an output before events
- WHEN contains the inputs and obligatory
- THEN contains the outputs and obligatory
- AND connects two GIVEN/WHEN/THEN test steps
- IS/ARE/DOES/DO/HAS/HAVE connects a category and a choice of this category, such as *MyCat IS MyChoice*

Keyword order in an AC

GIVEN

INITIALLY

WHEN

THEN

WHEN

WHEN

THEN

THEN

State transition testing becomes possible

Keywords example

GIVEN login IS successful   // PRECONDITION login IS successful

INITIALLY Number of short ticket IS 0 AND Number of normal ticket IS 0

WHEN Increase short IS #pressed

THEN Number of short ticket IS 1

WHEN Increase short IS #pressed

WHEN IT IS #pressed

THEN Number of short ticket IS 3

THEN Number of normal ticket IS 0

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/AC names.

Main menu (F): Select login; Exit

MyTest: GIVEN **Main menu** IS **Select login** WHEN total price IS 0 THEN paying IS not possible

YourTest: WHEN book price IS 50 Then Payment IS 45 WHEN **Main menu** IS **Exit**

For test step GIVEN no output will be generated

AC call without declaration

- We can call another test/AC without declaration in the categories

~~Main menu (F): Select login; Exit~~

MyTest: GIVEN **Main menu** IS **Select login** WHEN total price IS 0 THEN paying IS not possible

Difference – indirect call will not imply Select login to be involved in other test cases

Sub-acceptance criteria

- An AC, which can be used in other ACs.
- Common test steps can be used
- THEN is not required

SUB 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

- No test is generated from SUB ACs in their own features

Demo III – Gherkin++ tricks

Third exercise I. Buy water

Ordering water from an online shop.

R1. The types of water can be still or sparkling.

R2. You can buy bottles of one type only.

R3. If nothing is selected, then the quantity of either of them can be increased.

R4. After one has been selected, only one of the water types with a non-zero amount can be increased or decreased by one.

R5. The maximum number of bottles to be ordered is 10. If the selected number of bottles is greater than 0, then the buying process can start.

Make some acceptance criteria in th e following way:

1. make a PRECONDITION step using the category you think it's a real precondition
2. make an INITIALLY step using the category you think it's a real validation of an initial state
3. make an acceptance criterion which increases still to 10 and validates the final value 10. Use AND keyword when possible
4. use your previous acceptance criterion as sub AC to decrease still to 0 and validate the number of still for each step.
5. you can buy either still or sparkling water. Make a single acceptance criterion when you validate that you cannot buy sparkling if the number of still water is more than 0

Tables

Tables are used if acceptance criteria differ only in choices

```
Feature Table

original price(I): 20; 100; 1000
reduction(I): 10; 12; 0
payment method(I): card; money transfer
total price(O):  20; 90; 880


OrigPrice:
WHEN original price IS 20 | 100 | 1000
AND reduction IS 0 | 10 | 12
THEN total price IS 20 | 90 | 880
```

Tables – generated test cases

- Three test cases are generated
- Choices for Payment method alternate
- The choices of the first category should be different otherwise [#1], [#2] is used

OrigPrice [20]: WHEN original price IS 20 WHEN reduction IS 0 WHEN payment method IS card THEN total price IS 20

OrigPrice [100]: WHEN original price IS 100 WHEN reduction IS 10 WHEN payment method IS money transfer THEN total price IS 90

OrigPrice [1000]: WHEN original price IS 1000 WHEN reduction IS 12 WHEN payment method IS card THEN total price IS 880

Only

- ONLY + category names describe, which categories remain in a test case.
- **Single ONLY results in that only categories in the acceptance criterion are included in the test.**
- For sub-ACs the only-s are inherited and united

Only – example

Name(I): Smith; Wigner
Address(I): Fleet St; Millenium St
Payment(I): 4000; 5000
Title(I): Mr; Doctor

Happy path:
WHEN Name IS Smith AND Payment IS 5000

The test generated test case will be:

Happy path:
WHEN Name IS Smith
WHEN Address IS 12. Millenium St
WHEN Payment IS 5000
WHEN Title IS Mr

Only – example cont’d

Name(I): Smith; Wigner
Address(I): Fleet St; Millenium St
Payment(I): 4000; 5000
Title(I): Mr; Doctor

Happy path(ONLY):
WHEN Name IS Smith AND Payment IS 5000

The test generated test case will be:

Happy path:
WHEN Name IS Smith
WHEN Payment IS 5000

Collecting categories and choices: {}, and ?

- Lots of categories have the same choices such as #pressed or #present
- You can collect them by {}

{Exit; Go shopping; Log in; Log out; Register; Delete}? button (AO): #pressed; #present

WHEN Exit button IS #pressed
THEN Log in button IS #present
WHEN button IS # pressed

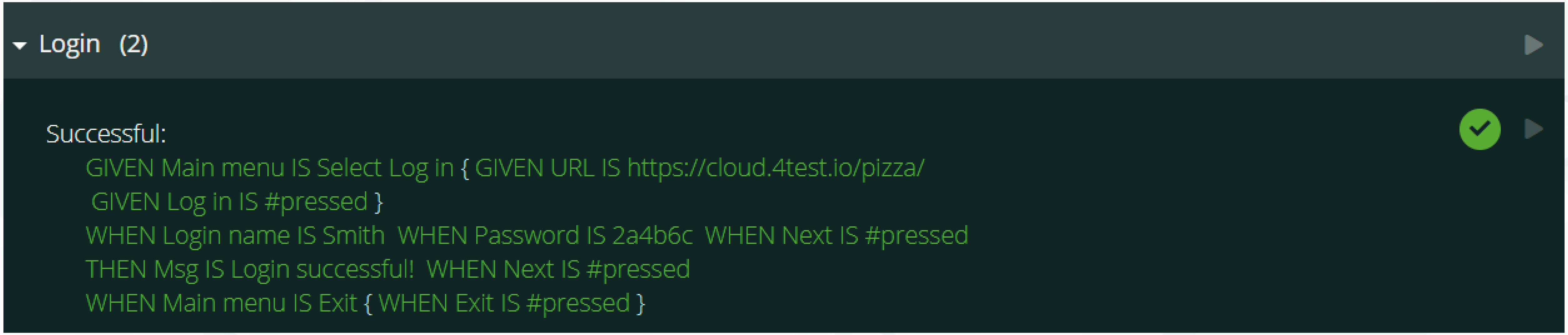
Fourth exercise

Make a single acceptance criterion in Login using table so that three test cases be generated:

1. Successful
2. Wrong log in name
3. Wrong password

Note that the only registered user is ‘Smith’ with password ‘2a4b6c’

Part III. Make your tests executable



Login (2)

Successful:

GIVEN Main menu IS Select Log in { GIVEN URL IS https://cloud.4test.io/pizza/
 GIVEN Log in IS #pressed }
WHEN Login name IS Smith WHEN Password IS 2a4b6c WHEN Next IS #pressed
THEN Msg IS Login successful! WHEN Next IS #pressed
WHEN Main menu IS Exit { WHEN Exit IS #pressed }

# for OKW, > for scope

Use # to set an event on the test object such as ‘pressed’, ‘active’, ‘non-present’, etc.

> is the scope operator for correct locators of similar GUI objects

Beer > Plus – Plus is in the scope of Beer, you can differentiate the identical Plus

#pressed – ClickOn

#present – VerifyExists: YES

#non-present – VerifyExists: NO

#active – VerifyIsActive: YES

#non-active – VerifyIsActive: NO

# for OKW, > for scope

WHEN Beer > Plus(A) IS #pressed

↓

SelectWindow("Beer");

ClickOn("Plus");

{Pizza Mexicana; Pizza Rucola XXL; Pizza Chicken; Coke; Beer} > {Plus; Delete} (A): #pressed

WHEN Pizza Mexicana > Plus IS #pressed
WHEN Coke > Plus IS #pressed
WHEN Pizza Mexicana > Delete IS #pressed

How to make your test cases executable

- Make only those ACs in a feature, which are related to the requirements of that feature.
- Start the application by using URL(I): https://cloud.4test.io/pizza/
- Make a precondition: PRECONDITION URL IS https://cloud.4test.io/pizza/
- Design your ACs to cover the requirements
- Complete the ACs with preconditions so that after the precondition has been executed, the program be in the entry point of the feature to be tested
- Make postconditions so that the next test can be correctly executed.
- Use ‘SUB’, ‘ONLY’, ‘>’ when necessary, and avoid starting the application more than once.

Example with demo IV.

Make ACs for the next additional requirement in Payment:
R10. After selecting items for at least 25 euros, the delivery is free. When an item is deleted but the total price still reaches 10 euros, the delivery fee becomes 3 euros.

Plan the ACs

- Make two SUB ACs in Main menu for shopping: (a) start the application (b) don't start the application
- Make a sub-AC shopping for 25 euros (Rucola, Mexicana, double coke)
- Make an AC based on the sub-AC by deleting cokes
- From payment:
    - call Main menu to start shopping (a),
    - call buy for 25,
    - validate free delivery,
    - call Main menu (b)
    - call deleting cokes,
    - validate delivery price

Troubleshooting

- Check your screen when the test fails.
- Reduce (cut) your test if necessary and execute a shorter test.
- The test steps maybe not correct. You can compare the actual screen where the test step failed with the test step in the AC.
- Missing ONLY. Harmony generates test steps for each input category, which sometimes superfluous. Use ONLY to filter out unnecessary categories with choices.
- Open URL more times. In this case, the application starts again and the necessary intermediate results lost.
- WHEN THEN swapping (usually a THEN is in the test step instead of WHEN)
- Restart Runner if something unpredictable happens.
- NO empty line or comment in a new line in an AC is permitted, this terminates the AC.

Fifth exercises

1. **Login**
2. **Register**
3. **Shopping**
4. **Payment**

The exercises are in the related features below the requirements

Thank you for participating in our workshop!

Let’s try Harmony on your projects!

Email: support@4testdev.com