TiD2016
质量竞争力大会
软件研发顶级盛会

LogiGear

SOFTWARE DEVELOPMENT
软件研发
更高 更新 更深

Test Design for Better Test Automation

汉斯. 布瓦达
Hans Buwalda
LogiGear
hans@logigear.com
@hansbuwalda

Who is your speaker?

LogiGear Corporation

- Silicon Valley and Vietnam
- Testing and test automation services:
  - consultancy, training
  - test development and automation services
  - "test integrated" development services
- Products:
  - TestArchitect™, TestArchitect for Visual Studio™
  - integrating test development with test management and automation

LogiGear®

www.logigear.com

Hans Buwalda

- Dutch guy, in California since 2001
- Background in math, computer science, management
- Focusing on keyword testing, agile testing, big testing

hans@logigear.com
@hansbuwalda

From The Netherlands (Holland)

Zwolle

Zwolle (兹沃勒)

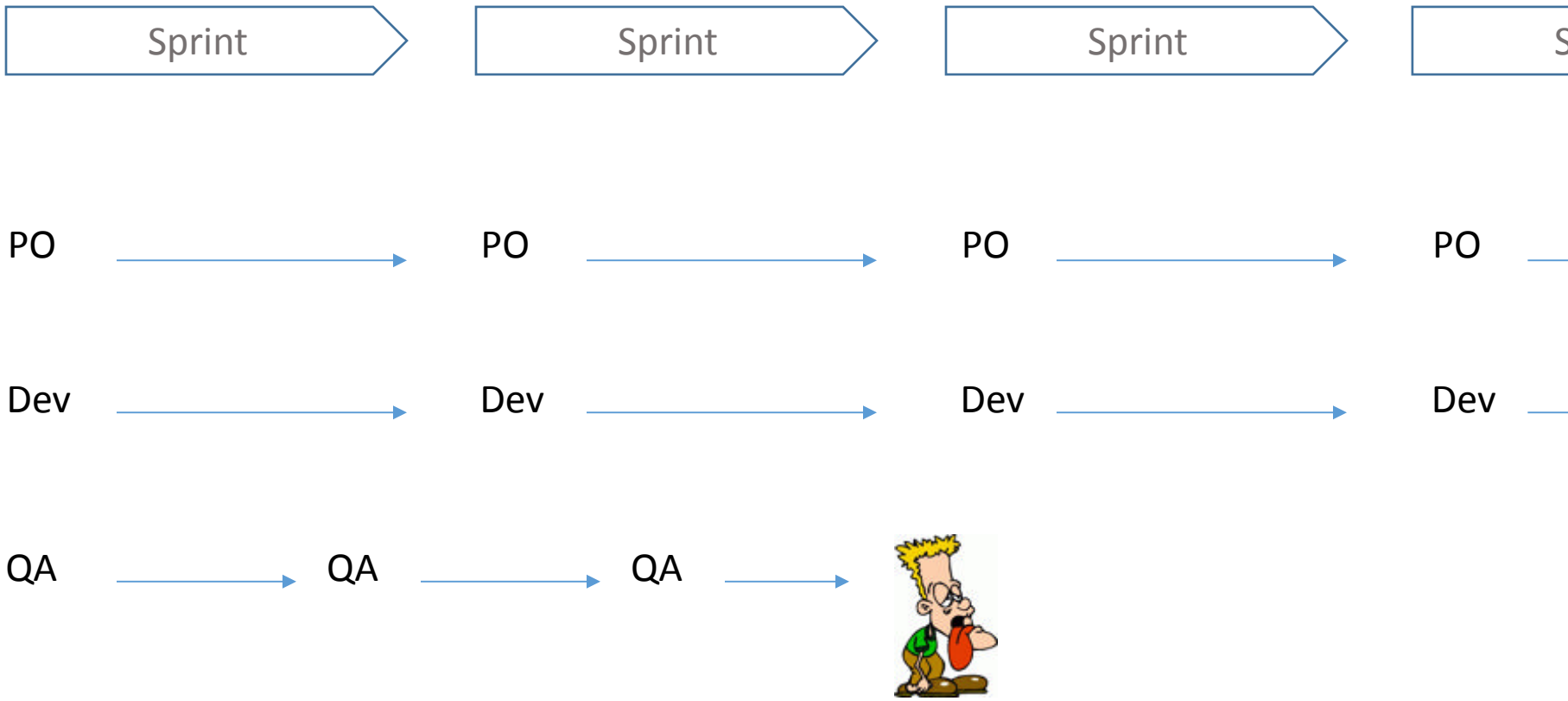
We like to ride bikes

America

Netherlands

Often hard to keep up for QA

SCRUM PROJECT



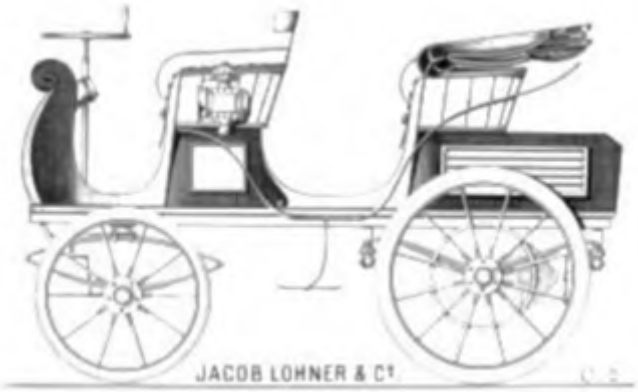
Cooperation in a team is important for QA. If that suffers QA will fall even further behind...

Tests and their scalability

	Relation to code	Quality / depth	Automation	Scalability
 **Unit Testing**	Close relationship with the code	Singular test scope, but deep into the code	Fully automated by nature	Scalable, grows with the code, easy to repeat
 **Functional Testing**	Usually does not have a one-on-one relation with code	Quality and scope depends on test design	In particular UI based automation can be a challenge	Often a bottle-neck in scalability
Exploratory Testing 	Human driven, not seeking a relation with code	Usually deep and thorough, good at finding problems	May or may not be automated afterwards	Not meant to be repeatable. Rather do a new session

Don't just automate manual testing

Don't just automate manual testing



Don't just automate manual testing

Good automated testing is not the same as automating good manual testing. . .

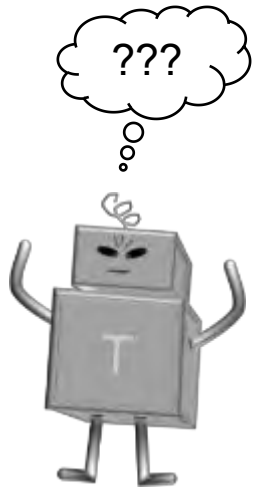
Example Scripting (Selenium)

```
WebElement element = driver.findElement(By.name(name));

element.sendKeys("mystery magic");
element.submit();

(new WebDriverWait(driver, 10)).until(
    new ExpectedCondition<Boolean>() {
        public Boolean apply(WebDriver d) {
            return d.getTitle()
            .toLowerCase().startsWith("mystery");
        }
    }
);

System.out.println(driver.getTitle());
```



Using Actions

fragment from a test with actions

	acc nr	first	last
open account	123123	John	Doe
	acc nr	amount	
deposit	123123	10.11	
deposit	123123	20.22	
	acc nr	expected	
check balance	123123	30.33	

4 actions, each with an action keyword and arguments

read from top to bottom

- Write the test as actions, in a spreadsheet
- Each action has a name and arguments
- Automate the actions, not the tests
- Each action is automated only once, easy to maintain

Variables and expressions with keywords

	car	available	
get quantity	Jeep	>> jeeps	
	first name	last name	car
rent car	John	Doe	Jeep
rent car	John	Doe	Jeep
	car	expected	
check quantity	Jeep	# jeeps - 2	

- Use ">>" to assign a value to a variable
- Use "#" to start an expression

Use existing actions to make new actions

- We use "enter", "click" and "check" to make a new action
- The new action is called "check balance"
- Arguments are used with "#"

define in one place:

ACTION DEFINITION	check balance		
	user		
argument	customer		
argument	amount		
	window	control	value
enter	balance inquiry	last name	# customer
	window	control	
click	balance inquiry	view balance	
	window	control	expected
check	balance inquiry	balance	# amount

use many times in tests:

	customer	amount
check balance	Smith	223.45
check balance	Jones	0.00
check balance	James	-330.45

3 ways to implement actions

- With action definitions
  - use existing actions to create new ones
  - see the previous slide
- Low level (UI, etc) actions can be built-in in the tool
  - like "click", "select menu item", etc
  - TestArchitect has 381 of such built-in actions
- Can program in a programming language
  - like C#, Python or Java

Example action implementation in Python

Script for an action check sort order, to verify whether the rows in a table are sorted:

```
def action_checkSortOrder():
    # get table object, column number and column count
    windowName = LIBRARY.NamedArgument("window")
    tableName = LIBRARY.NamedArgument("table")
    columnName = LIBRARY.NamedArgument("column")
    table = ABT.OpenElement(windowName, tableName)
    column = table.GetColumnIndex(columnName)
    rowCount = table.GetRowCount()

    # check the sort order, row by row
    previous = table.GetCellText(0, column)
    for i in range(1, rowCount):
        current = table.GetCellText(i, column)
        if  current < previous :
            LIBRARY.AdministerCheck("order", "sorted", "fails " + str(i+1), 0)
            return
        previous = current

    LIBRARY.AdministerCheck("order", "sorted", "all rows in order", 1)
```

get arguments from the test line

find the table in the UI

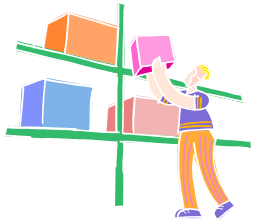
if a value is smaller than before, fail the test

if all rows are ascending, pass the test

Using the new action

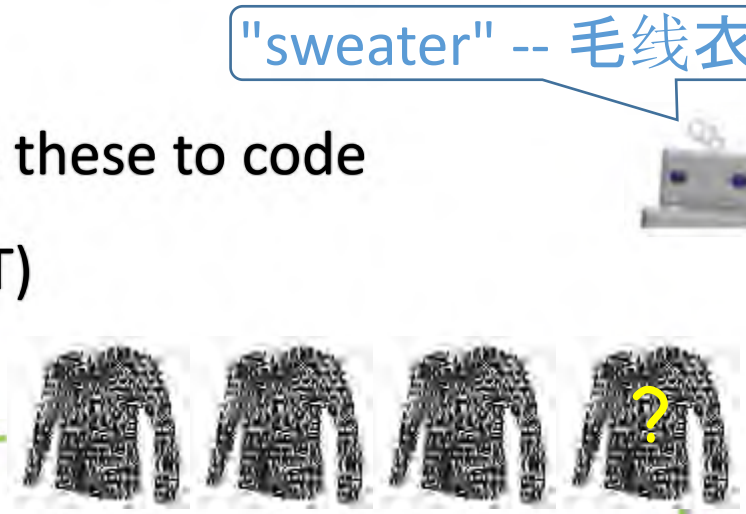
- By keeping an action generic it can be applied for a variety of situations
- Some examples of using "check sort order":

	window	table	column
check sort order	view orders	orders table	ID
check sort order	annual results	regions	revenue
check sort order	inventory	cars	price
check sort order	registration	students	last name



Behavior Driven Development (BDD)

- Uses natural language scenarios
- Tools like JBehave and Cucumber map these to code
- Structure is "Given-When-Then" (GWT)
- Example:



Given a customer previously bought a black sweater from me

And I currently have three black sweaters left in stock

When he returns the sweater for a refund

Then I should have four black sweaters in stock

(source: Wikipedia)

BDD with actions (example 1)

GIVEN	A customer previously bought a black sweater from me		
	first	last	id
add client	John	Jones	>> client
AND	I currently have three black sweaters left in stock		
	article	color	quantity
get stock	sweater	black	>> sweaters
WHEN	He returns the sweater for a refund		
	customer	article	color
return article	# client	sweater	black
THEN	I should have four black sweaters in stock		
	article	color	quantity
check stock	sweaters	black	# sweaters + 1

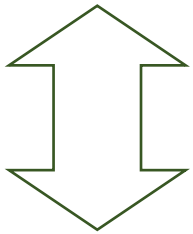
BDD with Actions (example 2)

Given a customer previously bought a black sweater from me

And I currently have three black sweaters left in stock

When he returns the sweater for a refund

Then I should have four black sweaters in stock



```
customer buys, article, color
    a customer previously bought a {color} {article} from me

set stock, article, color, amount
    I currently have {amount} {color} {article} left in stock
    my stock of {color} {sweater} is {amount}

return article, article, color
    he returns the {color} {article} for a refund

check stock, article, color
    I should have four {color} {article} in stock
```

	article	color	
customer buys	sweater	black	
	article	color	amount
set stock	sweater	black	3
	article	color	
return article	sweater	black	
	article	color	amount
check stock	sweater	black	4

Risks of keyword approaches

- Keywords are not a magic solution to all problems

- With many tests and keywords maintenance can still be hard

The method I will show you today:

"Action Based Testing"

My statement on test automation . . .

Successful automation is not as much a technical challenge, as it is a test design challenge.

Issues are not always obvious...

Downton Abbey (唐顿庄园)

Why Better Test Design?

Quality
will be better

⇒

Automation
will be easier

Example of a Test

is this a good test? is it good for automation?

Description	Expected
Open http://www.bigstore.com	The "BIG Store" main page is displayed, with a "sign in" link
Click on "Sign In", upper right corner	A sign in dialog shows, the "Sign in" button is disabled
Enter "johnd" in the user name field	The "Sign In" button is still disabled
Enter "bigtester" in the password field	Now the "Sign In" button is enabled
Click on the "Sign in" button	The page now shows "Hello John" in the upper right corner
Enter "acme watch" in the search field	The "Search" button is enabled
Click on the "Search" button	5 watches of Acme Corporation are displayed
Double click on "Acme Super Watch 2"	The details page of the Acme Super Watch 2 is displayed
Verify the picture of the watch	The picture should show a black Acme Super Watch 2
Select "red" in the "Color" dropdown list	The picture now shows a red Acme Super Watch 2
Type 2 in the "Order quantity" textbox	The price in the right shows "$79.00 + Free Shipping"
Click on "Add to cart"	The status panel shows "Acme Super Watch 2" added
Click on "Check out"	The Cart Check Out open, with the 2 Acme Super Watches
etc	etc

BDD scenario . . .

Given User turns off Password required option for Drive Test
And User has logged in by Traffic Applicant account
And User is at the Assessments Take a Test page
And User clicks the Traffic Test link
And User clicks the Next button
And User clicks the Sheet radio button in Mode page if displayed
And User clicks the Start button
And User waits for test start
And User clicks the Stop Test button
When User clicks the Confirm Stop Test button
And User enters the correct applicant password
And User clicks the Confirm Stop Test button
Then The Test is Over should be displayed in the Message label
And the value of the Message label should be The test is over
And The Welcome to Traffic Testing page should be displayed

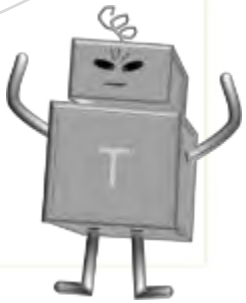
The Test Automation Design Paradox

悖论

Test design is important for automation

but . . .

many testers are not engineers



Test Module

Objectives

Initial - setup

Test cases

Final - cleanup

TEST MODULE	Car Rental Reservation			
OBJECTIVES				
test objective	TO 01	Invoice total is calculated correctly		
test objective	TO 02	Number of currently available cars is updated after a rental reservation		
test objective	TO 03	An error message is displayed when user tries to rent unavailable car		
INITIAL	Setting up			
config				
start application				
login				
TEST CASE	TC 01	Check invoice total for a single reservation		
test objective	TO 01	Invoice total is calculated correctly		
	first name	last name	car	rental option
rent car	John	Doe	Compact/Toyota Prius	Infant Seat ($10.0/day)
	last name	total charge		
check total payment	Doe	140.40		
TEST CASE	TC 02	Check invoice total for multiple reservations		
TEST CASE	TC 03	Check number of currently available cars is updated		
FINAL	Cleaning up			
	current window			
back to home	car selection			
	window			
close application	welcome			

(1) Early in the project: identify the test modules

tester, engineers and product owners can do this together

these become the closet with labeled (empty) shelves that the tests will be stored in later

Price Calculations

Login Tests

Address Entry

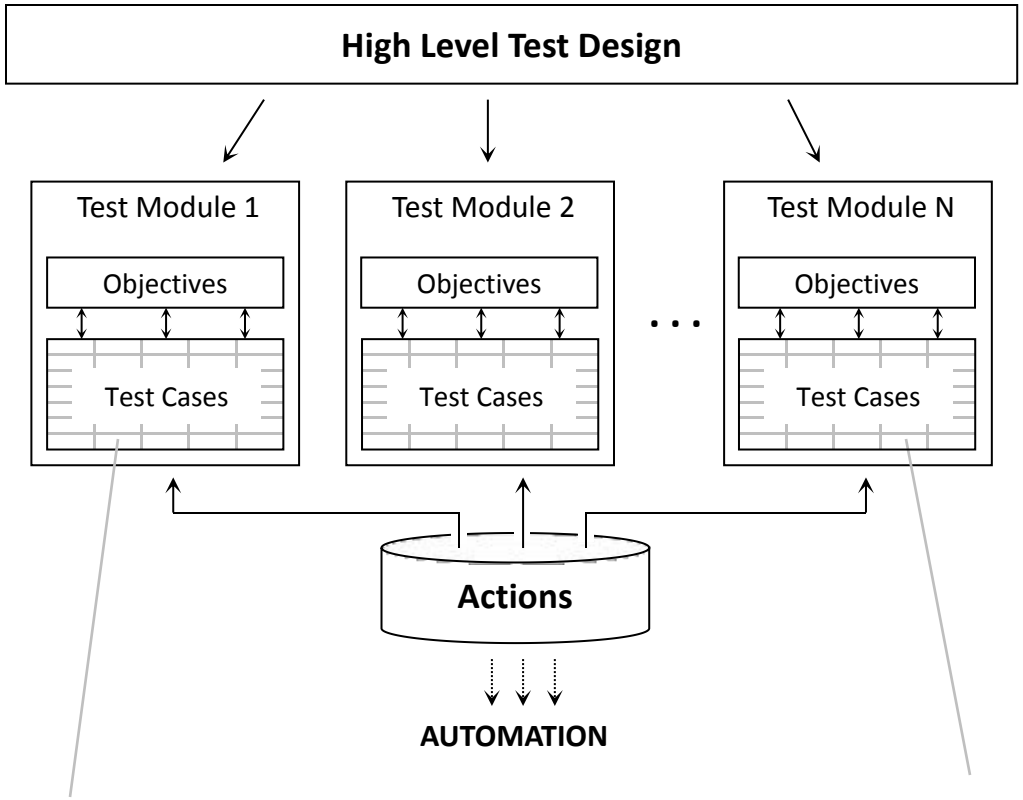
(2) During the project: fit the tests into the modules

only use actions and checks that fit the scope of the module

avoid unnecessary details, hide them in the actions

but don't hide too much, info that is needed to understand a test should be visible

Overview Action Based Testing



define the "chapters"

create the "chapters"

create the "words"

make the words work

interaction test

	window	control	value	
enter	log in	user name	jdoe	
enter	log in	password	car guy	
	window	control	property	expected
check property	log in	ok button	enabled	true

business test

	user	password		
log in	jdoe	car guy		
	first	last	brand	model
enter rental	Mary	Renter	Ford	Escape
	last	total		
check bill	Renter	140.42		

Breakdown Criteria

- Common Criteria
  - Functionality (customers, finances, management information, UI, ...)
  - Architecture of the system under test (client, server, protocol, sub systems, components, modules, ...)
  - Kind of test (navigation flow, negative tests, response time, ...)

- Additional Criteria
  - Stakeholders (like "Accounting", "Compliance", "HR", ...)
  - Complexity of the test (put complex tests in separate modules)
  - Execution aspects (special hardware, multi-station, ...)
  - Project planning (availability of information, timelines, sprints, ...)
  - Risks involved (extra test modules for high risk areas)
  - Ambition level (smoke test, regression, aggressive, …)

Eye on the ball, Scope

- Always know the scope of the test module
- The scope should be clear and unambiguous
- The scope determines many things:
  - what the test objectives are
  - which test cases to expect
  - what level of actions to use
  - what the checks are about and which events should generate a warning or error (if a “lower” functionality is wrong)

Use the right level actions

Hide details if they don't matter (don't fit the scope of the test):

	window	tree	tree item path
click tree item	main	projects	/Project/Drill Assembly
	window	list	item
check list item exists	main	tasks	Plan of Approach

⇩ ⇩ ⇩

	project	task
check task in project	Drill Assembly	Plan of Approach

however . . .

. . . show details if they do matter. This line for example is unclear:

	user name	old pw	short pw
check invalid userid pwd	johnson	abcdef99	abc

???

Tip: have default values for arguments

ACTION DEFINITION		login	
	name	*default value*	
argument	user	tester	
argument	password	testerpw	
	window	*control*	*value*
enter	login window	user name	# user
enter	login window	password	# password
	window	*control*	
click	login window	login	

Sometimes explicit values are needed:

	user	*password*
login	tamaraj	tj1234
	text	
check message	Hello Tamara	

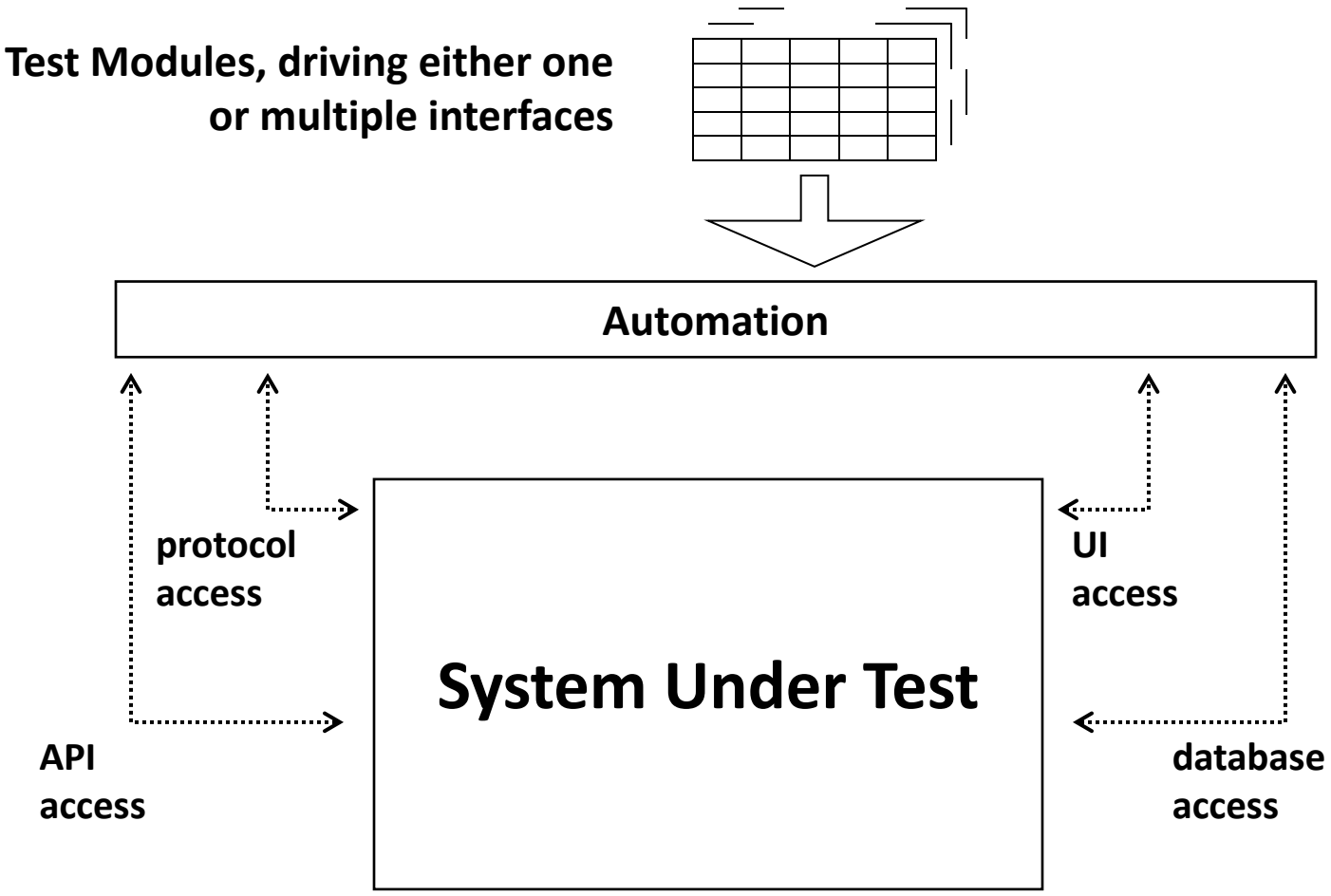
Usually default values are good enough:

login			
	date	*payee*	*amount*
make payment	1/1/12	Gas Co.	85.00

Non-UI

- Examples
  - web services (REST and SOAP)
  - application programming interfaces (API's)
  - embedded software
  - protocols
  - files, batches
  - databases, SQL
  - command line interfaces (CLI's)
  - multi-media
- Can be target of tests, and/or automation interfaces
- Non-UI automation can speed up functional tests that do not address the UI
  - but it can also complicate them

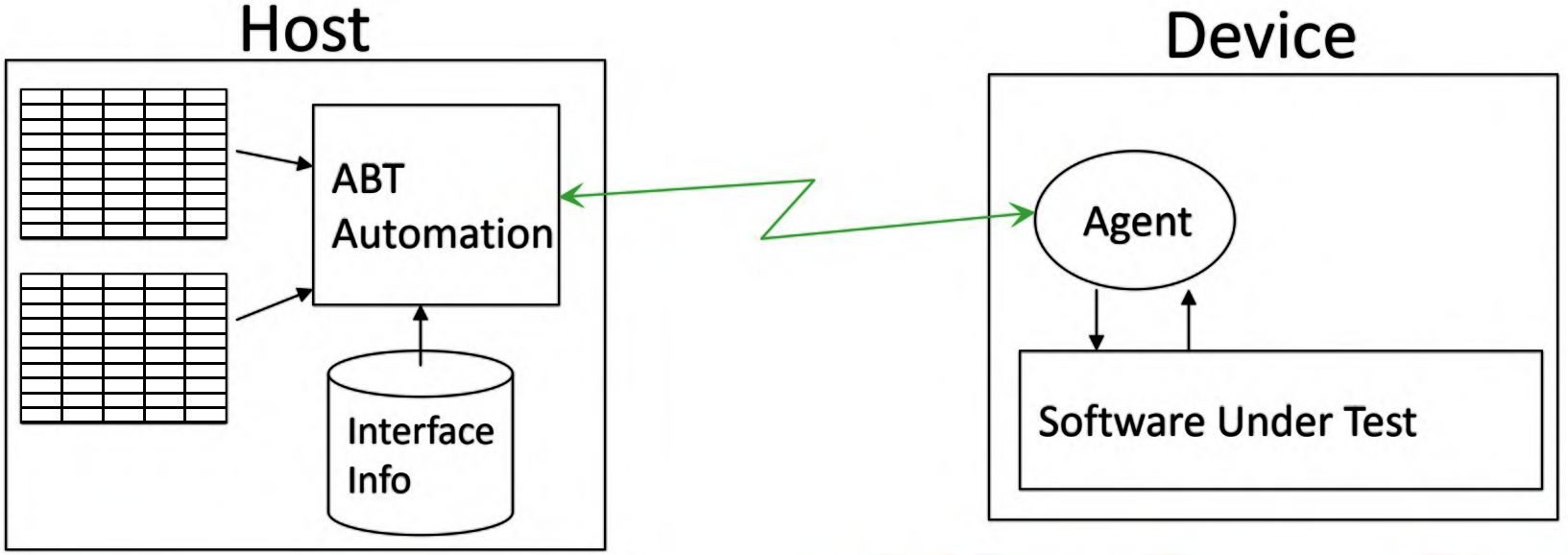
Multiple System Access



Example approach: using an agent

do as much as possible here

do as little as possible here



Mobile Devices

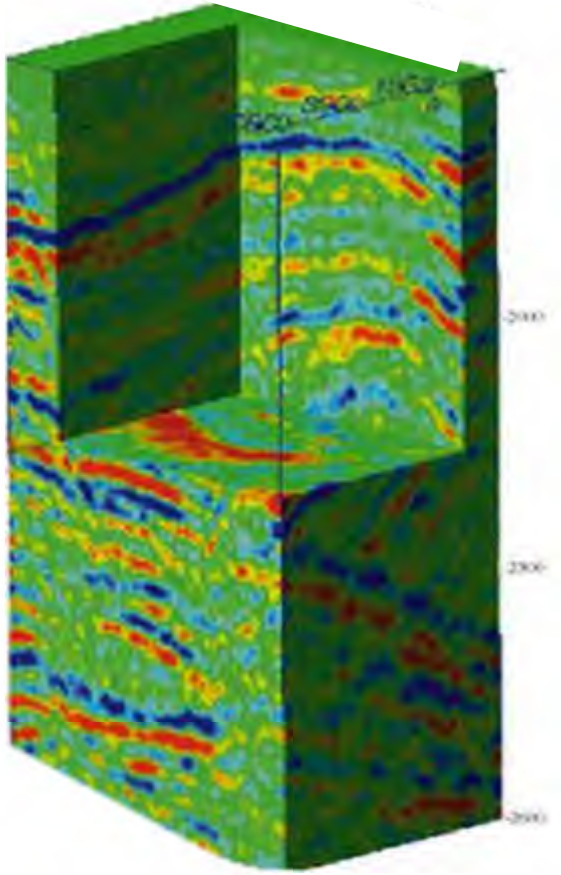
Car Diagnostic Systems

Games, VR

Point of Sale Terminals

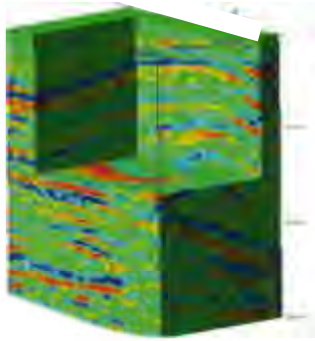
Multimedia: The "check picture" Approach

- Approach applicable for pictures like graphics or icons
- The tester will add a line "check picture", that includes a "question" as one of the arguments
- While the test is executed TA keeps the recorded pictures
- After execution the pictures are shown to a manual testing for approval
- Once approved unchanged same pictures won't be presented to the manual tester again in future runs

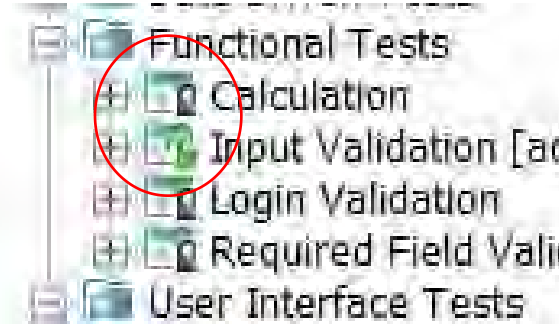


Absolute and relative picture checks

- Relative picture checks are part of test module
- They depict images that are specific for that module
- Example a chart based on calculations. In another module the chart will typically be different (based on a different calculation)



- Absolute picture checks refer to pictures that occur in multiple tests, based on conditions, but always the same image
- Their names start with "/"
- Typical use: icons or pictograms:
  - as a tester you usually don't care about the design of an icon, but whether it appears when expected



Data driven testing

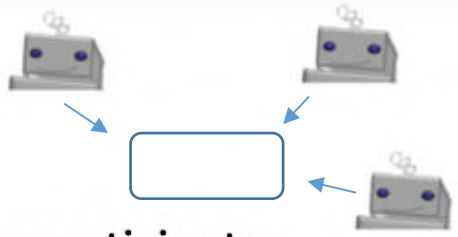
TEST CASE	TC 03	Check stocks
	data set	
use data set	/cars	
	car	available
get quantity	# car	>> quantity
	first name	last name car
rent car	# first	# last # car
	car	expected
check quantity	# car	# quantity - 1
repeat for data set		

DATA SET cars

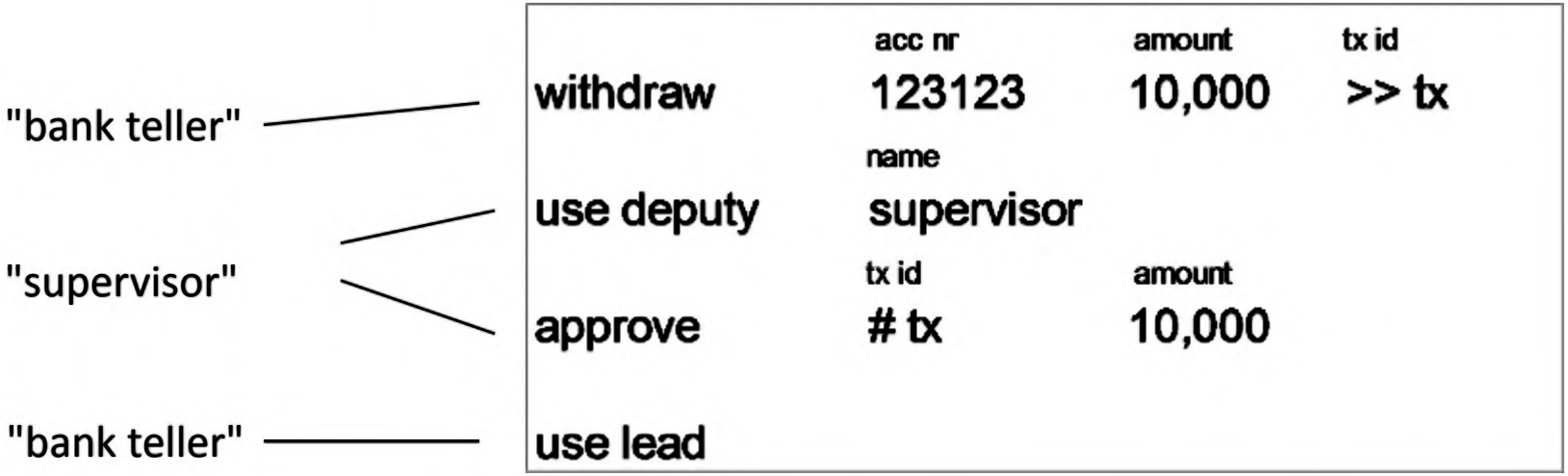
first	last	car
John	Doe	Chevy Volt
Mary	Kane	Ford Escape
Jane	Collins	Chrysler 300
Tom	Anderson	Buick Verano
Henry	Smyth	BMW 750
Vivian	Major	Toyota Corolla

- The test lines will be repeated for each row in the data set
- The values represented by "car", "first" and "last" come from the selected row of the data set
- The data set can be static (in a table) or dynamic (generated at run time)

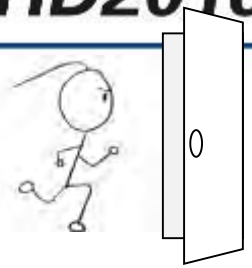
"Lead Deputy" Testing



- For "multi station" testing, when multiple machines have to participate in a test in real-time
- For example if a supervisor needs to approve a withdrawal in a bank teller system
- Can be "sync" and "parallel" (with a rendezvous point)



Takeaways



- Test design is a major contributor to automation success, often more than technical expertise
- Domain language approaches like Action Based Testing and BDD allow for efficient communication and driving of automation
- Test modules can help organize the tests, and focus their scopes
- Focusing tests, checks and actions on a clear and differentiated scope will make for better tests, but also better automation

Thank you, my email is: hans@logigear.com