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

How to make large and complex testing projects successful

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

Who is here?

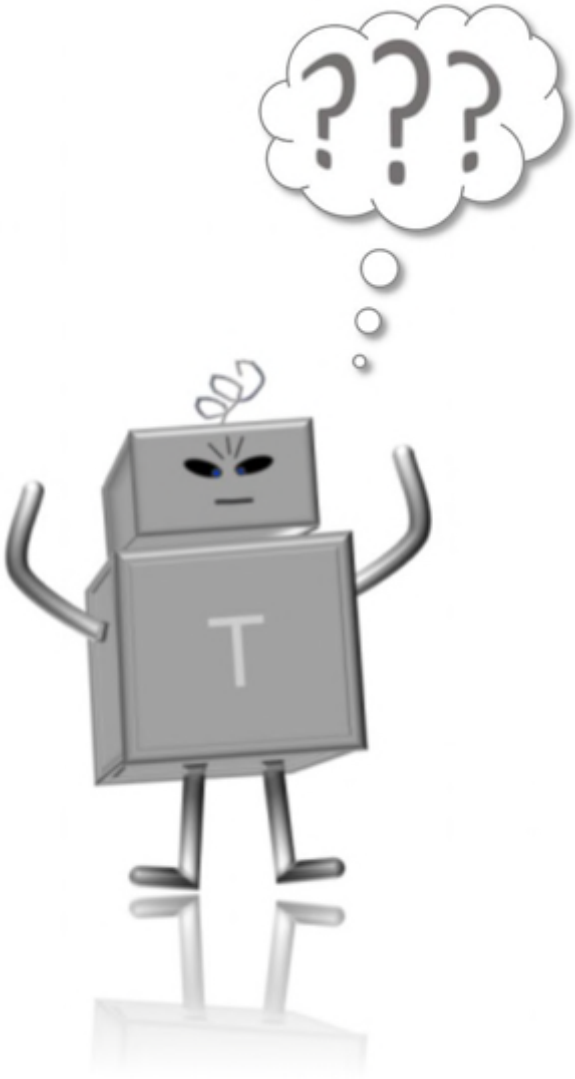
- Industries, roles
- How good is your English? ☺

Key items in scalable automation

- Organization and design of the tests
- Process and cooperation (agile or traditional)
- Project and production focus
- Technology
- Infrastructure
- Testability of the application under test
- Agreement, commitment
- Globalization, off-shoring

Existential Questions

- Why test?
- Why not test?
- Why automate tests?
- Why not automate tests?



Why test?

- People expect us to do
- Somebody wants us to
- Increases certainty and control
  - Showing absence of problems ("Looking for good news")
- Finds faults, saving time, money, damage
  - Showing presence of problems ("looking for bad news")

Why not test?

- It costs time and money
- You might find problems . . .
- We forgot to plan for it
- We need the resources for development
- It is difficult
- It's hard to manage

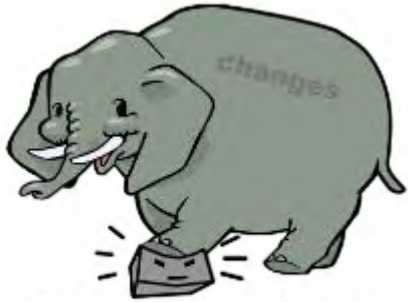
Why Automate Tests?

- It is more fun
- Can save time and money
  - potentially improving time-to-market, quality-to-market and control
- Can capture key domain and application knowledge
- Can speed up development life cycles
- Execution (执行) typically is more reliable
  - a robot is not subjective, tired or moody
- Some tests can be done much better, or only, with automation

Why not Automate?

- Can rule out the human elements
  - promotes "mechanical" testing
  - might not find "unexpected" problems
- More sensitive to good practices (作法)
  - pitfalls are plentiful
- Needs technical expertise in the test team
- Tends to dominate the testing process
  - at the cost of good test development
- Creates more software to manage
  - can actually diminish scalability rather than helping it
  - in particular changes in an application under test can have large, and hard to predict, impact on the automated tests

"Almost 50% of Agile teams can't automate more than 29% of their tests, and half of those can't automate more than 10%.
Non-agile is even less."
-- Forester 2016



changes in an application can crush automation...

What are we talking about today?

	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

Action Based Testing

Using Actions - 用行动

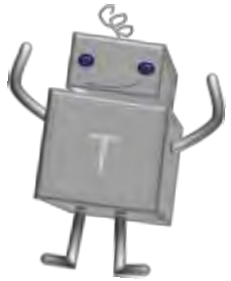
fragment from a test with actions

	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

"deposit" -- 存款
"balance" -- 结余



- 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

Organize actions in "Test Modules"

- A test module consists of an (1) initial part, (2) test cases and (3) a final part
- Focus is on readability and a clear scope
- Navigation details are avoided, unless they're meant to be tested

TEST MODULE	Car Rental Payments			
	user			
start system	jdoe			
TEST CASE	**TC 01**	**Rent some cars**		
	first name	last name	car	days
rent car	Mary	Jones	Ford Escape	3
	first name	last name	amount	
check billing	Mary	Jones	140.40	
FINAL				
close application				

"rent car" means renting a car, like: 租一辆自驾车

"billing" is the American word to tell the customer how much to pay, like in: 帐单

Variables and expressions

TEST CASE	TC 02	Rent some more cars	
	car	available	
get quantity	Chevy Volt	>> volt	
	first name	last name	car
rent car	John	Green	Chevy Volt
rent car	Jane	White	Chevy Volt
	car	expected	
check quantity	Chevy Volt	# volt - 2	

- This test does not need an absolute number for the available cars, just wants to see if a stock is updated
- As a convention we keep a value in a variable with ">>"
- The "#" indicates an expression (calculation like 表达式)

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

Test Module

Objectives

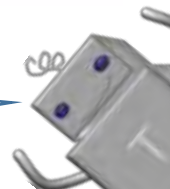
Initial - setup

Test cases

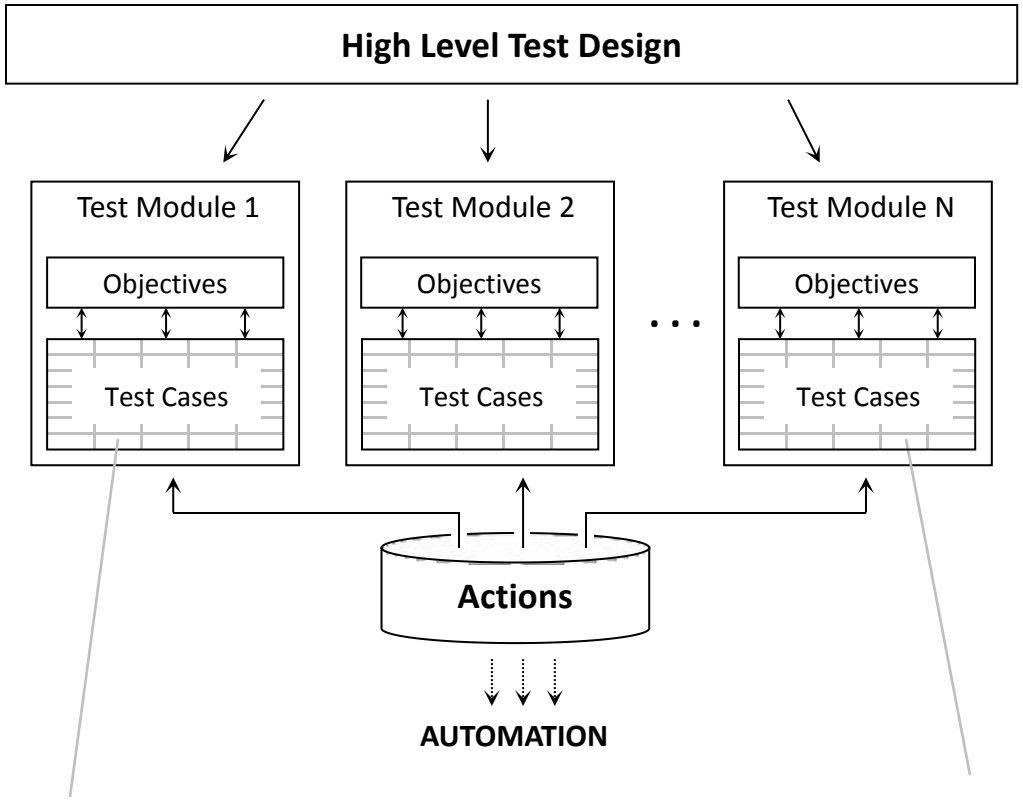
Final - cleanup

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			

objective" -- 目的



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		

Why Better Test Design?

- Quality of test
  - many tests are often quite "mechanical" (机械) now, no surprises
  - one to one related to specifications, user stories or requirements, which often is ok, but lacks aggression
  - no combinations, no unexpected situations, lame and boring
  - such tests have a hard time finding (interesting) bugs
- Better automation
  - when unneeded details are left out of tests, those details don't have to be maintained
  - limit the impact of system changes on tests, making such impact more manageable
  - bottom line: good test design makes automated testing more scalable

I have become to believe that successful automation is usually less of a technical challenge as it is a test design challenge.

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

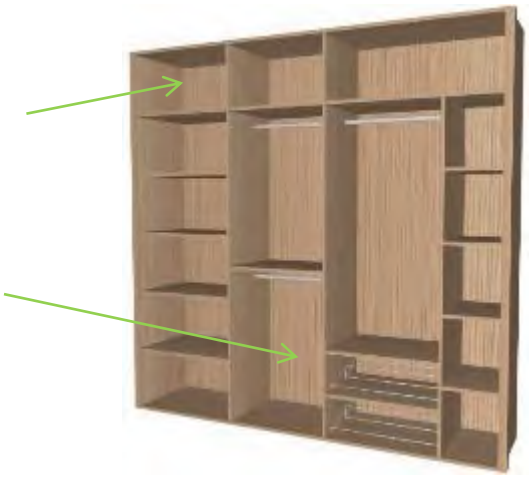
- By keeping an action generic it can be applied for a variety of situations
- Some examples of using "check sort order":

	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

Cleaning up a garage (车库)

What's the secret (秘诀) . . .

- Get facilities to store and organize your content
- Select your stuff
- Decide where to put what
  - assign and label the shelves
- Put it there

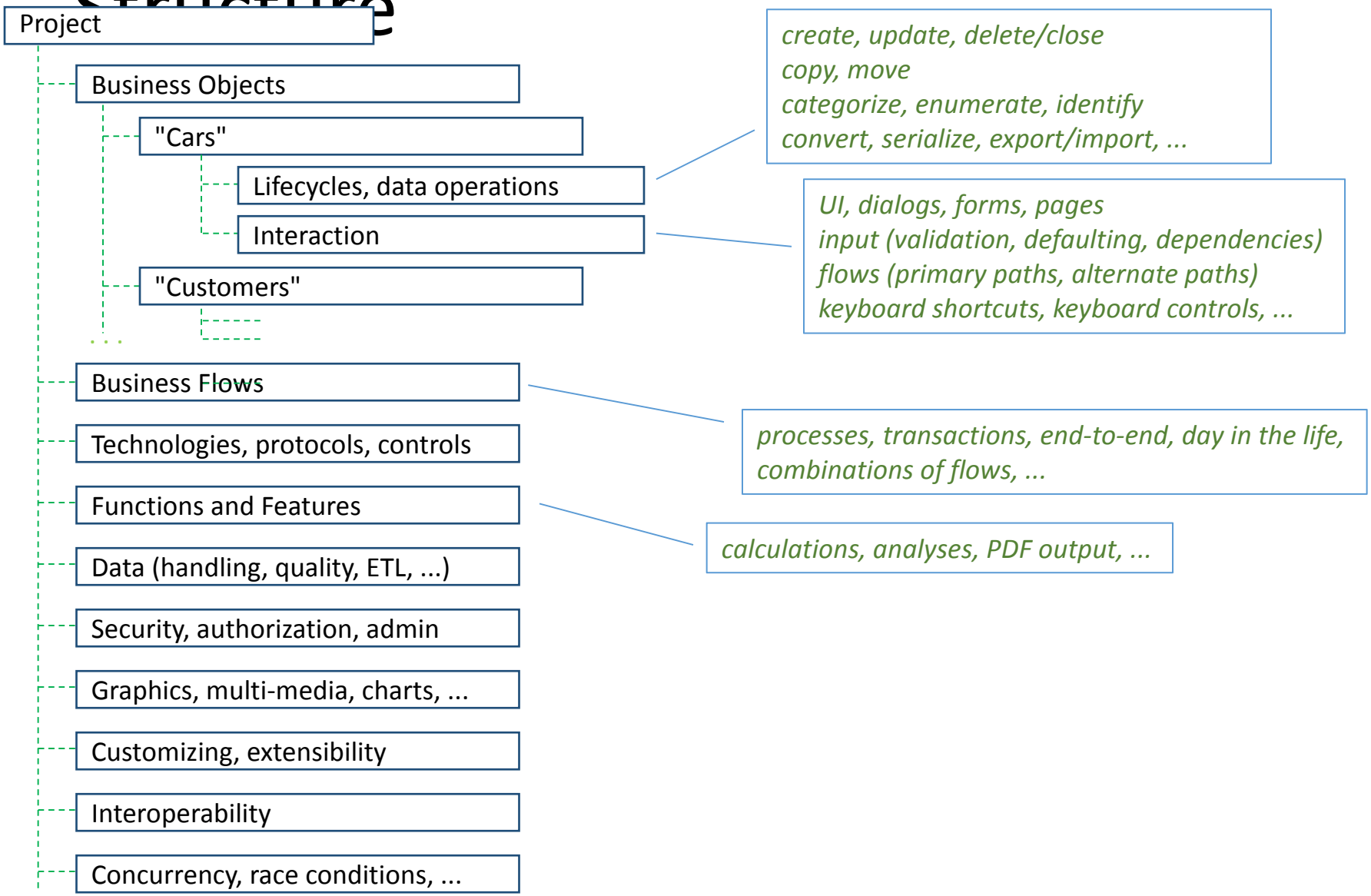


- If the organization is not sufficient anymore, add to it or change it

Examples of considerations

- Business objects and business flows
  - objects are like cars, invoices, locations, etc
  - flows are like create, fulfill, pay and close an order
- Other tests
  - functions and features, like premium calculation or PDF output
  - administration, users, security, authorizations
  - graphics
  - technologies, protocols, ...
  - customization, extensibility
  - interoperability
  - . . .
- Business versus interaction
  - differentiate within business objects and all other categories
  - interaction can be further differentiated into: values, UI, keyboard, etc
  - also, for every category, look at negative tests, flows, aggressive tests, etc

Example Top Level Structure



Example: E-commerce site

- Articles
- Categories
- Customers
- Promotions
- Orders
- Invoices
- Payments
- Credit cards
- Staff members
- Countries
- . . .

Promotions -- business tests

- Promotion kinds
  - percentage discounts
  - fixed dollar discounts
  - comprehensive promotions
  - regional promotions
  - volume discounts
  - prompt payment discounts
  - article based, comprehensive
  - . . .
- Life cycles
  - creation
  - modification
    - various properties
    - before or after activation
  - expiration
  - cancellation

Promotions -- interaction tests

- Screen by screen
- Discount percentage/fixed
  - does the amount field change
- Regional checkbox
  - does the region list show up
  - are the regions correct
  - can the regions be selected/deselected
- Start, duration, end date
  - are the date checks working
  - can either duration or end date be entered
- Articles, categories
  - can categories be selected, do they show up
  - do the right articles show up, can they be selected
- . . .

What tests could look like

Business tests for business object "Promotions"

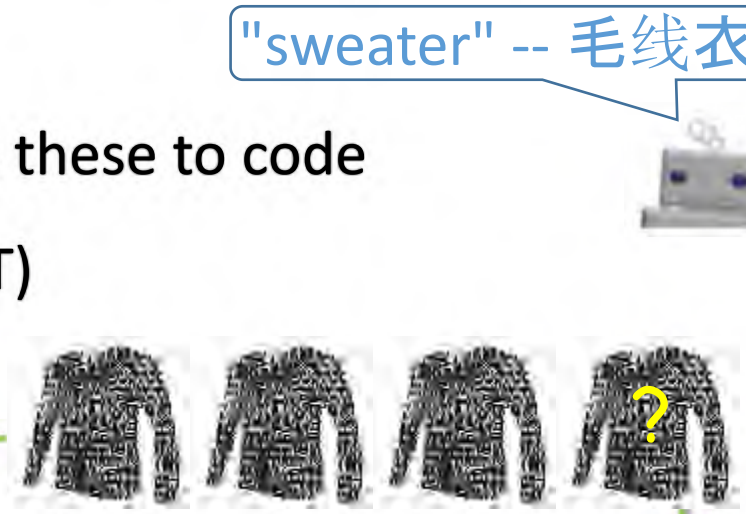
	name	start	finish	percentage	category
create promotion	christmas	12/1/2016	12/25/2016	11	all
create promotion	tablets	12/20/2016	1/1/2017	20	tablets
	date				
time travel	12/23/2016				
	article	price			
check nett price	iPad Mini 4	284.79			

Interaction tests for business object "Promotions"

	window	control	
click	main	new promotion	
	window	type	
select	promotion	town	
	window	list	value
check list item exists	promotion	towns	Tietjerksteradeel

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 action(example)

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

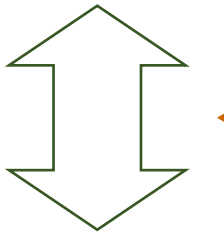
Equivalence, conversion

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	amount
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

Test Development

Approach per Test Module

- Plan the test module:
  - when to develop: do we have enough information?
    ⇨ UI tests are usually the last ones to be developed
  - when to execute: make sure lower level stuff working first
    ⇨ UI tests are usually the first ones to be executed
- Process:
  - do an intake: understand what is needed and devise an approach
  - analyze requirements, formulate "test objectives", create tests
- Identify stakeholders and their involvement:
  - users, subject matter experts
  - developers
  - auditors
- Choose testing techniques if applicable:
  - boundary analysis, decision tables, etc
- Try to follow an exploratory approach:
  - see the test development as a "learning process", about the business domain, the application structure, the interaction, etc
  - talk about your tests, make them strong

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)

Example of a Test

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

Step	Description	Expected
step 16	Open http://www.bigstore.com	The "BIG Store" main page is displayed, with a "sign in" link
step 17	Click on "Sign In", upper right corner	A sign in dialog shows, the "Sign in" button is disabled
step 18	Enter "johnd" in the user name field	The "Sign In" button is still disabled
step 19	Enter "bigtester" in the password field	Now the "Sign In" button is enabled
step 20	Click on the "Sign in" button	The page now shows "Hello John" in the upper right corner
step 21	Enter "acme watch" in the search field	The "Search" button is enabled
step 22	Click on the "Search" button	5 watches of Acme Corporation are displayed
step 23	Double click on "Acme Super Watch 2"	The details page of the Acme Super Watch 2 is displayed
step 24	Verify the picture of the watch	The picture should show a black Acme Super Watch 2
step 25	Select "red" in the "Color" dropdown list	The picture now shows a black Acme Super Watch 2
step 26	Type 2 in the "Order quantity" textbox	The price in the right shows "$79.00 + Free Shipping"
step 27	Click on "Add to cart"	The status panel shows "Acme Super Watch 2" added
step 28	Click on "Check out"	The Cart Check Out open, with the 2 Acme Super Watches

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

Use of "Anti-patterns"

- Informal way to classify typical test design problems
- Use with care, can come across offensive
- The point of view in the following list is automation: test design choices that can be counter-productive to automation
- However, lack of good organization can also effect the quality of manual tests

See my article:
Techwell Insights
September 17, 2015

"Anti-patterns" (informal)

- *Interaction Heavy* – Not having many business tests
- *Lame* – No depth or variety, no testing techniques used
- *Enter Enter Click Click* – Test steps are too detailed
- *No Life* – Missing life cycle steps of business objects
- *Clueless* – No clear scope for the tests
- *Cocktail* – Interaction tests mixed with business tests
- *Over-Checking* – Checks not relevant for the scope
- *Sneaky Checking* – Checks hidden in actions
- *Action Explosion* – Many actions, hardly different
- *Mystery Actions* – Cryptic actions, unclear what they do
- *Techno* – Actions and tests that look like code
  - often _NOts0EasY_2REad
  - but great to impress non-technical people
- *Sleepy* – Use of hard sleeps, risking speed stability
- *Black box* – Lack of testability support in the AUT
- *Varimania* – More use of variables than the test actually needs

Dealing with data

- Constructed data is easier to manage
  - can use automation to generate it, and to enter it in the environment
  - result of test analysis and design, reflecting "interesting" situations
  - however, less "surprises": real life situations which were not foreseen
- Real-world data is challenging to organize
  - make it a project, or task, in itself
  - make absolutely sure to deal with privacy, security and legal aspects appropriately. You may need to "scrub" the data
- Consider using automation to select data for a test
  - set criteria ("need a male older than 50, married, living in Denver"), query for matching cases, and select one randomly (if possible a different one each run)
  - this approach will introduce variation and unexpectedness, making automated tests stronger and more interesting

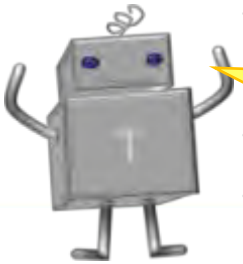
Designing Actions

Actions

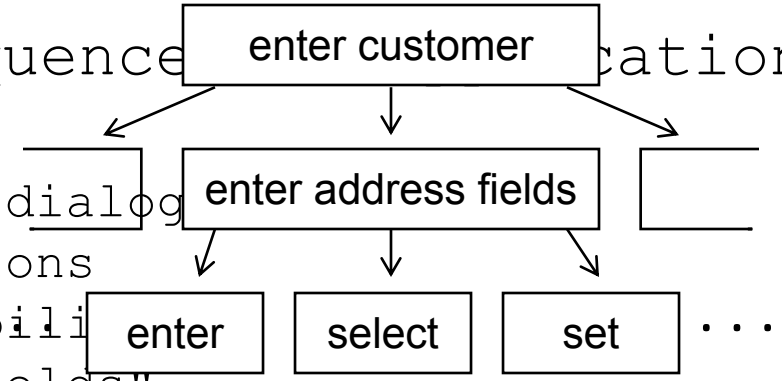
- By product of test design
- As generic as possible
- Use a verb and a noun, and standardize the verbs and the nouns
  - verb: "check" or "verify" ?
  - noun: "customer" or "client" ?
  - so a new action will be "check customer" or "verify client" ?
- Organize and document

Low-level, high-level, mid-level actions

- "Low level": detailed interaction with the UI (or API)
  - examples: "click", "expand tree node", "select menu"
- "High level": a business domain operation or check on the application under test
  - hide the interaction
  - examples: "enter customer", "rent car", "check balance"
- "Mid level": common sequence[illegible]cation level
  - usually to wrap a form or dialog[illegible] actions
  - [illegible]ainabili[illegible]
  - example: "enter address fields"



tip: do not click confirmation buttons like "ok", "next", "submit" etc in a mid-level action



Tip: Provide default values in actions

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	

Use login action w/ arguments

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

Use login default values

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

Using actions

TEST MODULE	Order processing				
start system					
TEST CASE	**TC 01**	**Order for tablets**			
	user	password			
login	jdoe	doedoe			
	window				
check window exists	welcome				
	order id	cust id	article	price	quantity
create order	AB123	W3454X	tablet	198.95	5
	order id	total			
check order total	AB123	994.75			
. . .					

Some notes on Bugs

Bugs found in the "Immediate Sphere", when the developer/team is still working on the code (like in the same sprint)

⇨ Consider not logging as bugs, since that is much overhead.
Simply share a failed test module with the developer.

Bugs found "Post Delivery", when the developer/team is working on something else already

⇨ Good to keep track, manage, prioritize, assign, close, learn etc.
The later the bug is found the more important.

⇨ For each bug found late ask three questions, in this order:

1. was it a bug?
2. what was the root cause?
3. why wasn't it caught?

Consider keeping this information in the tracking system

Varying Tests

Vary your tests?

- Automated tests have a tendency to be rigid, and predictable
- Real-world situations are not necessarily predictable
- Whenever possible try to vary:
  - with select other data cases that still fit the goal of tests
  - with randomized behavior of the test

Generation and randomization techniques

- Model-based
  - use models of the system under test to create tests
  - see: Harry Robinson, www.model-based-testing.org, and Hans Buwalda, Better Software, March 2003
- Data driven testing
  - apply one test scenario to multiple data elements
  - either coming from a file or produce by an automation
- "Monkey testing"
  - use automation to generate random data or behavior
  - "smart monkeys" will follow typical user behavior, most helpful in efficiency
  - "dumb monkeys" are more purely random, may find more unexpected issues
  - long simulations can expose bugs traditional tests won't find
- Extended Random Regression
  - have a large database of tests
  - randomly select and run them, for a very long time
  - this will expose bugs otherwise hidden
  - see Cem Kaner e.a.: "High Volume Test Automation", STARWEST 2004

Data driven testing with keywords

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

Combinations

- Input values
  - determine equivalence classes of values for a variable or field
  - for each class pick a value (or randomize)
- Options, settings
- Configurations
  - operating systems, operating system versions and flavors
    - Windows service packs, Linux distributions
  - browsers, browser versions
  - protocol stacks (IPv4, IPv6, USB, ...)
  - processors
  - DBMS's
- Combinations of all of the above
- Trying all combinations will spin out of control quickly

Pairwise versus exhaustive testing

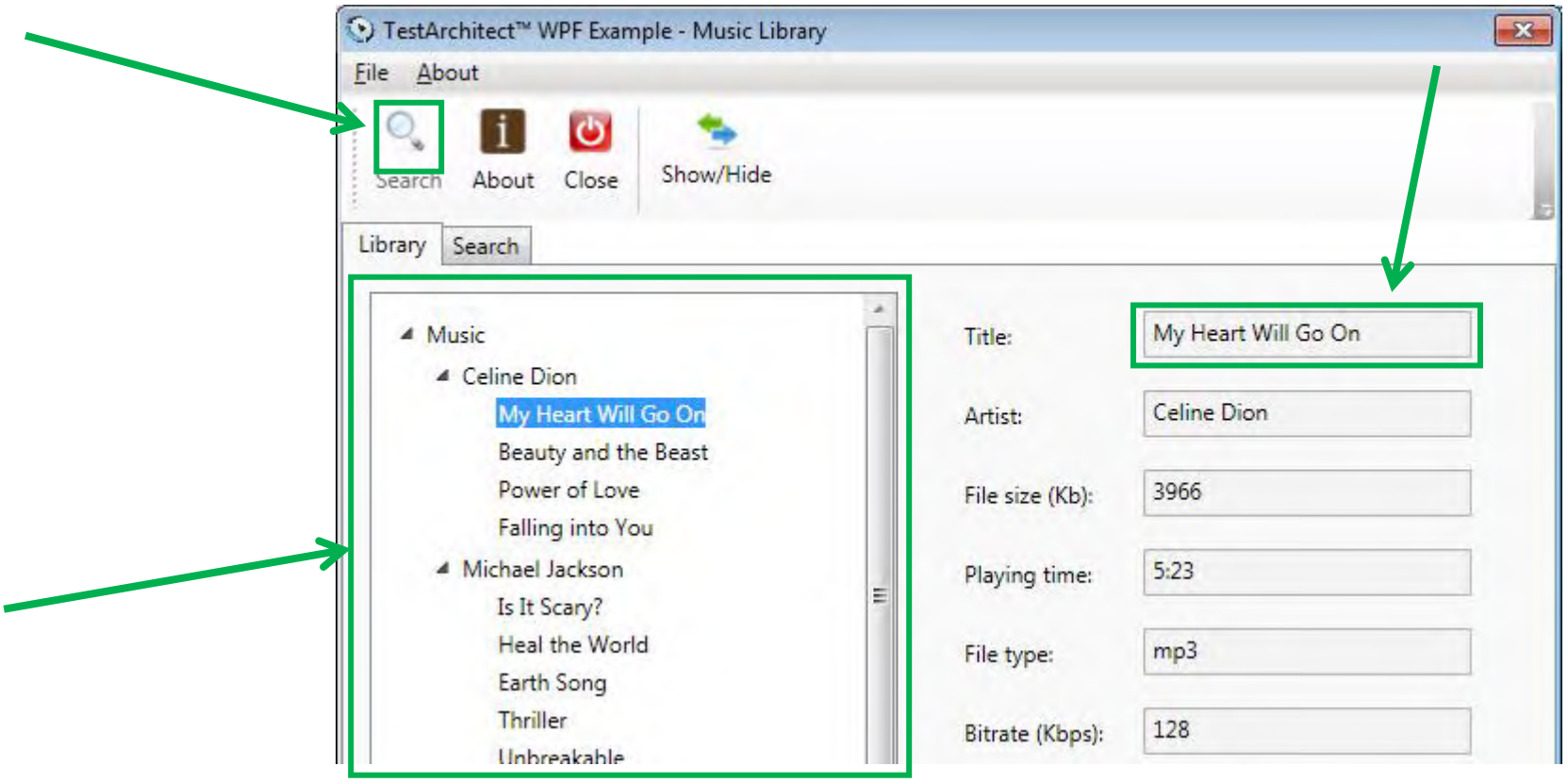
- Group values of variables in pairs (or tuples with more than 2)
- Each pair (tuple) should occur in the test at least once
  - maybe not in every run, but at least once before you assume "done"
  - consider to go through combinations round-robin, for example pick a different combination every time you run a build acceptance test
- Example, configurations
  - operating system: Windows XP, Apple OS X, Red Hat Enterprise Linux
  - browser: Internet Explorer, Firefox, Chrome
  - processor: Intel, AMD
  - database: MySQL, Sybase, Oracle
  - 72 combinations possible, to test each pair: 10 tests
- Example of tools:
  - ACTS by NIST
  - PICT by Microsoft,
  - AllPairs by James Bach (Perl)
  - for a longer list see: www.pairwise.org

Test	OS	Browser	Protocol	CPU	DBMS
1	XP	IE	IPv4	Intel	MySQL
2	XP	Firefox	IPv6	AMD	Sybase
3	XP	IE	IPv6	Intel	Oracle
4	OS X	Firefox	IPv4	AMD	MySQL
5	OS X	IE	IPv4	Intel	Sybase
6	OS X	Firefox	IPv4	Intel	Oracle
7	RHEL	IE	IPv6	AMD	MySQL
8	RHEL	Firefox	IPv4	Intel	Sybase
9	RHEL	Firefox	IPv4	AMD	Oracle
10	OS X	Firefox	IPv6	AMD	Oracle

Source: PRACTICAL COMBINATORIAL TESTING, D. Richard Kuhn, Raghu N. Kacker, Yu Lei, NIST Special Publication 800-142, October, 2010

Technical notes

Identifying controls



- Identify windows and controls, and assign names to them
- These names encapsulate the properties that the tool can use to identify the windows and controls when executing the tests

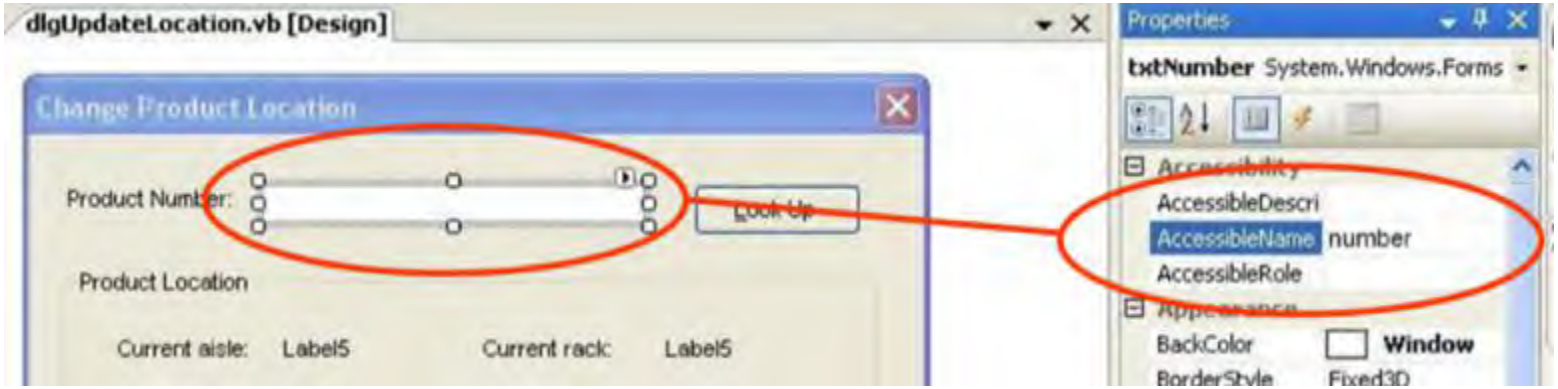
Mapping an interface

INTERFACE ENTITY	library		
interface entity setting	title	{.*Music Library}	
	name	class	label
interface element	title	text	Title:
interface element	artist	text	Artist:
interface element	file size	text	File size (Kb):
	name	class	position
interface element	playing time	text	textbox 4
interface element	file type	text	textbox 5
interface element	bitrate	text	textbox 6
	name	class	position
interface element	music	treeview	treeview 1

- An interface mapping (common in test tools) will map windows and controls to names
- When the interface of an application changes, you only have to update this in one place
- The interface mapping is a key step in your automation success, allocate time to design it well, in particular naming and choosing identifying properties

Hidden interface properties

- Use properties a human user can't see, but a test tool can
- This approach can lead to speedier and more stable automation
  - less need for "spy" tools (which take a lot of time)
  - less sensitive to changes in the system under test
  - not sensitive to languages and localizations
- A "white-box" approach to UI's can also help operate on or verify aspect of interface elements
- Examples:
  - "id" attribute for HTML elements
  - "name" field for Java controls
  - "AccessibleName" or "Automation ID" properties in .Net controls (see below)



Using the hidden identifiers

INTERFACE ENTITY	library		
interface entity setting	automation id	MusicLibraryWindow	
	ta name	ta class	automation id
interface element	title	text	TitleTextBox
interface element	artist	text	SongArtistTextBox
interface element	file size	text	SizeTextBox
interface element	playing time	text	TimeTextBox
interface element	file type	text	TypeTextBox
interface element	bitrate	text	BitrateTextBox
	ta name	ta class	automation id
interface element	music	treeview	MusicTreeView

- Instead of positions or language dependent labels, an internal property "automation id" has been used
- The interface definition will be less dependent on modifications in the UI of the application under test
- If the information can be agreed upon with the developers, for example in an agile team, it can be entered (or pasted) manually and early on

Active Timing

- Passive timing
  - wait a set amount of time
  - in large scale testing, try to avoid passive timing altogether:
    - if wait too short, test will be interrupted
    - if wait too long, time is wasted
- Active timing
  - wait for a measurable event
  - usually the wait is up to a, generous, maximum time
  - common example: wait for a window or control to appear (usually the test tool will do this for you)
- Even if not obvious, find something to wait for...
- Involve developers if needed
  - relatively easy in an agile team, but also in traditional projects, give this priority
- If using a waiting loop
  - make sure to use a "sleep" function in each cycle that frees up the processor (giving the AUT time to respond)
  - wait for an end time, rather then a set amount of cycles

Things to wait for...

- Wait for a last control or elements to load
  - developers can help knowing which one that is
- Non-UI criteria
  - API function
  - existence of a file
- Criteria added in development specifically for this purpose, like:
  - "disabling" big slow controls (like lists or trees) until they're done loading
  - API functions or UI window or control properties
- Use a "delta" approach:
  - every wait cycle, test if there was a change; if no change, assume that the loading time is over:
  - examples of changes:
    - the controls on a window
    - count of items in a list
    - size a file (like a log file)

Testability, key items

- Should be a "must have" requirement
  - first question in a development project: "how do we test this?"
- Design of the system(s) under test:
  - components, tiers, services,
- Hidden identifying properties
- Hooks for timing
- White-box access to anything relevant:
  - input data (ability to emulate)
  - output data (what is underlying data being displayed)
  - random generators (can I set a seed?)
  - states (like in a game)
  - objects displayed (like monsters in a game)
- Emulation features, like time-travel and fake locations

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
  - mobile devices

Alexander Graham Bell
"Device Testing"

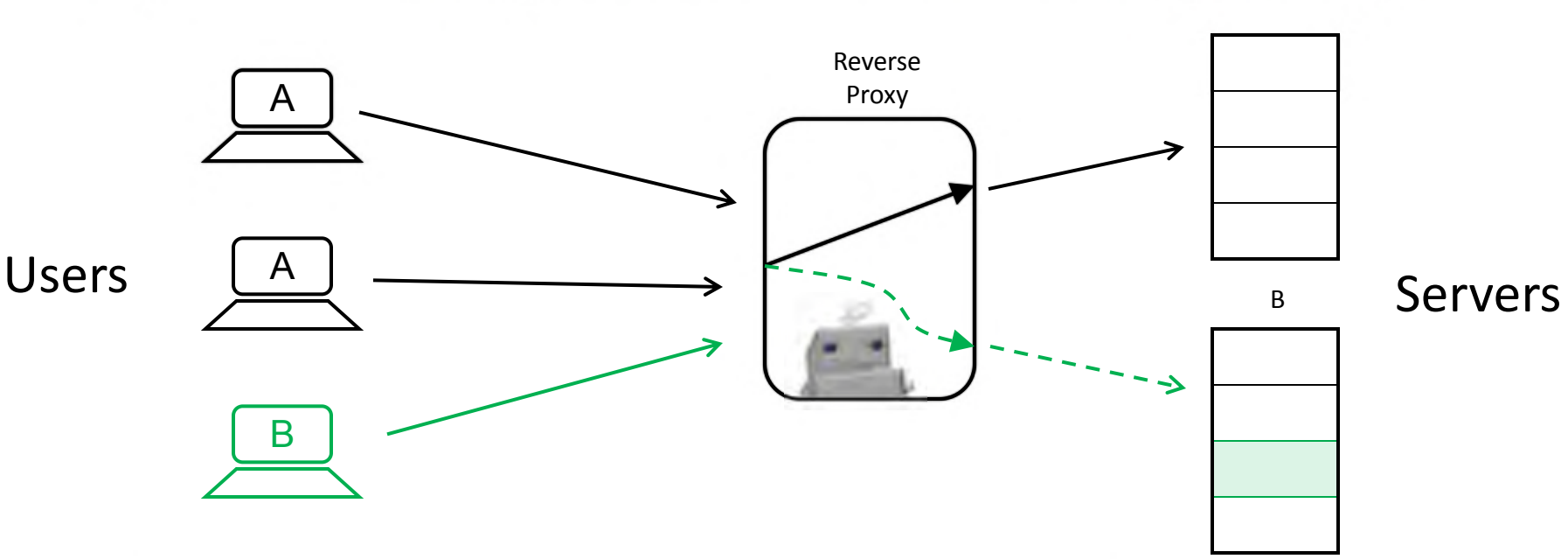
- Can be target of tests, and/or automation interfaces
- Non-UI automation can speed up functional tests that do not address the UI
  - but it can also complicate them

Testing in, or near, production

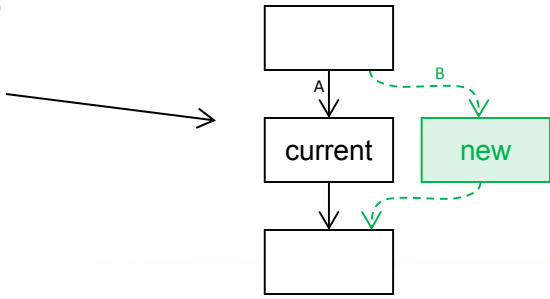
- In an agile and DevOps approach continuous deployment is becoming the norm
  - in particular for service based system (like web, apps, SaaS, but also client-server)
  - logic in services is easier to manage and updated than distributed and variable clients
- In a DevOps approach tight cooperation between development, test and deployment pays off
  - automation testability
  - testable system designs
  - good test design remains a success factor
  - tight integration in the build, deployment and production processes
- A/B testing can help test vary complex systems
  - used not only for testing, also for trying out new features
  - dividing the incoming traffic into an A and B flow (B is the new situation)
  - automated tests can use the B flow
  - to do this, have it well integrated in your system designs
- Continuous testing with random regression testing or monkey testing

*see also: Ken Johnston's chapter in the book of Dorothy Graham and Mark Fewster, and his keynote at StarWest 2012

A/B testing with a reverse proxy



- A/B testing means part of traffic is routed through a different server or component (see if it works, and/or how users react)
- B could be a real-life user, a test user or also an automated test
- The strategy can be done at any component level
- Watch your test design, easy to drown in technical only

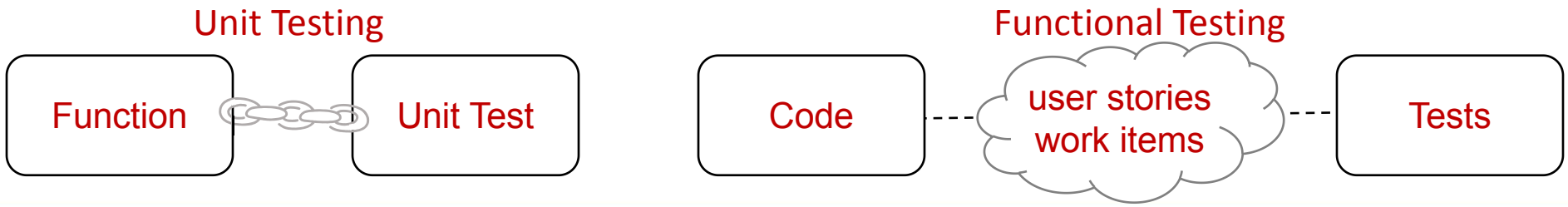


New stuff . . .

- Identify and address new or unusual testing and automation challenges, before they might delay teams

Test Execution

- Have an explicit approach for when and how to execute which tests
  - a good high level test design will help with this
- Execution can be selective or integral
  - unit tests are typically executed selectively, often automatically based on code changes in a system like SVN or TFS
  - functional tests don't have as obvious relations with code files
  - selective execution will be quicker and more efficient, integral execution may catch more side-effect issues ("bonus bugs")
  - consider "random regression" execution of tests



Virtualization

虚拟化

Virtualization, a testers dream...

- In particular for functional testing
- Much easier to define and create needed configurations
  - you basically just need storage
  - managing this is your next challenge
- One stored configuration can be re-used over and over again
- The VM can always start "fresh", in particular with
  - fresh base data (either server or client)
  - specified state, for example to repeat a particular problematic automation situation
- Can take "snap shots" of situations, for analysis of problems
- Can use automation itself to select and start/stop suitable VM's
  - for example using actions for this
  - or letting an overnight or continuous build take care of this

Virtualization, bad dream?

- In many situations testers have no control over their VM's
  - allocated by an IT department, or hiding in a cloud
- Capacity
  - automated testing is a lot more demanding than human testing
- Virtual machine latency can add timing problems
  - timing might be off if a VM is not running continuously
  - we see this quite often derailing big test runs
- Management of images
  - images can be (very) large, and difficult to store and move around,
  - and their number can grow exponentially we configuration needs
  - distinguish between managed configurations and sandboxes
  - define ownership, organize it
  - IT may be the one giving out (running) VM's, restricting your flexibility

Cloud

云端运算

Cloud models (summary)

Amazon profit crushes estimates as cloud-service revenue soars

Alibaba Joins With SoftBank For Japanese Cloud Computing Expansion

- Infrastructure as a service (IaaS)
  - renting (virtual) machines
  - you maintain images with your stuff in it
  - this is handy for testing:
    - can maintain and use multiple configurations
    - can get testing done quickly by temporarily allocating machines
- Platform as a service (PaaS)
  - images are predefined (OS, DBMS, etc)
  - advantages for testing are similar to IaaS
- Software as a service (SaaS)
  - the application is web-based
  - the cloud manages usage and load balancing
  - testing strategy is different:
    - less need for different configurations (on the server side)
    - may include testing in production strategies like A/B testing

Cloud for testing

- Cloud can be target of testing
  - usually SaaP applications
  - typically a mix of UI and service testing
  - from multiple locations (can be another use of the cloud)
- Cloud can be host of test execution
  - the test client will be in the cloud, with a copy of the application, or with a browser
  - can generate specific testing situations, or just speed up testing
  - make sure machines are rented and returned efficiently
- Amazon is the market leader,
  - Microsoft is pushing Azure very hard
  - well known Chinese players are Ali Baba (Ali Yun) and Baidu (Baidu Yun)
- Public cloud providers offer API's, so your automation can automatically allocate and release them
  - be careful, software bugs can have costing consequences
  - for example, consider having a second automation process to double-check cloud machines have been released after a set time

Public cloud use needs organization

- You're spending money, therefore decide who can do what (don't forget to limit you yourself too)
- Have a "test production planning" process
- Have a budget (预算)
- Have ownership (所有权)
- Use available policy features to limit usage in time and quantity
- Obtain and read production reporting, compare to plan and budget
- Minimize the need (for example "last test round only")
- Have and try to use on-prem and hybrid alternatives
- Start small, learn

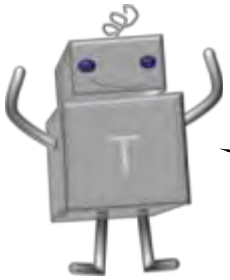
Organization, Process

Organization

- Much of the success is gained or lost in how you organize the process
  - who owns which responsibility (in particular to say "no" to a release)
  - separate, integrated teams, or both
  - who does test design, who does automation
  - what to outsource, what to keep in-house

- Write a plan of approach for the test development and automation
  - scope, assumptions, risks, planning
  - methods, best practices
  - tools, technologies, architecture
  - stake holders, including roles and processes for input and approvals
  - team
  - . . .

- Assemble the right resources
  - testers, lead testers
  - automation engineer(s)
  - managers, diplomats, ...



Test design is a skill . . .
Automation is a skill . . .
Management is a skill . . .

. . . and those skills are different

Testing as a profession (专业)

- From the ISTQB report on World Wide Testing Practices:
  - close to 80% of cases testers are in a separate organization
  - in 84% of cases the test team does not report to Development
- Focus on tests, not development:
  - what can be consequences of situations and events
  - relieve developers
- The challenge (挑战) for the tester in the new era is to be a credible professional (可信的专业)
  - not a pseudo programmer (不是伪程序员)
  - part of the team
  - have knowledge and experience with testing techniques and principles
  - know the domain
- Forcing a nontechnical tester to become a programmer may lose a good tester and gain a poor programmer
- Forcing a good developer to become a tester may lose a good developer and gain a poor tester
  - a good developer who is working on an airplane control system is also not necessarily a good airline pilot

Automation is a profession too

- Not the same as regular system development
- Navigating (导航) through other software efficiently (有效率), dealing with control classes, obtaining information, timing, etc
  - if you would compare developers to "creators", automation engineers might be likened to "adventurers" (or "puzzlers") . . .
- The automation engineering role can also be a consultant:
  - for test developers: help express tests efficiently
  - for system developers: how to make a system more automation friendly
  - important player in innovation in the automated testing

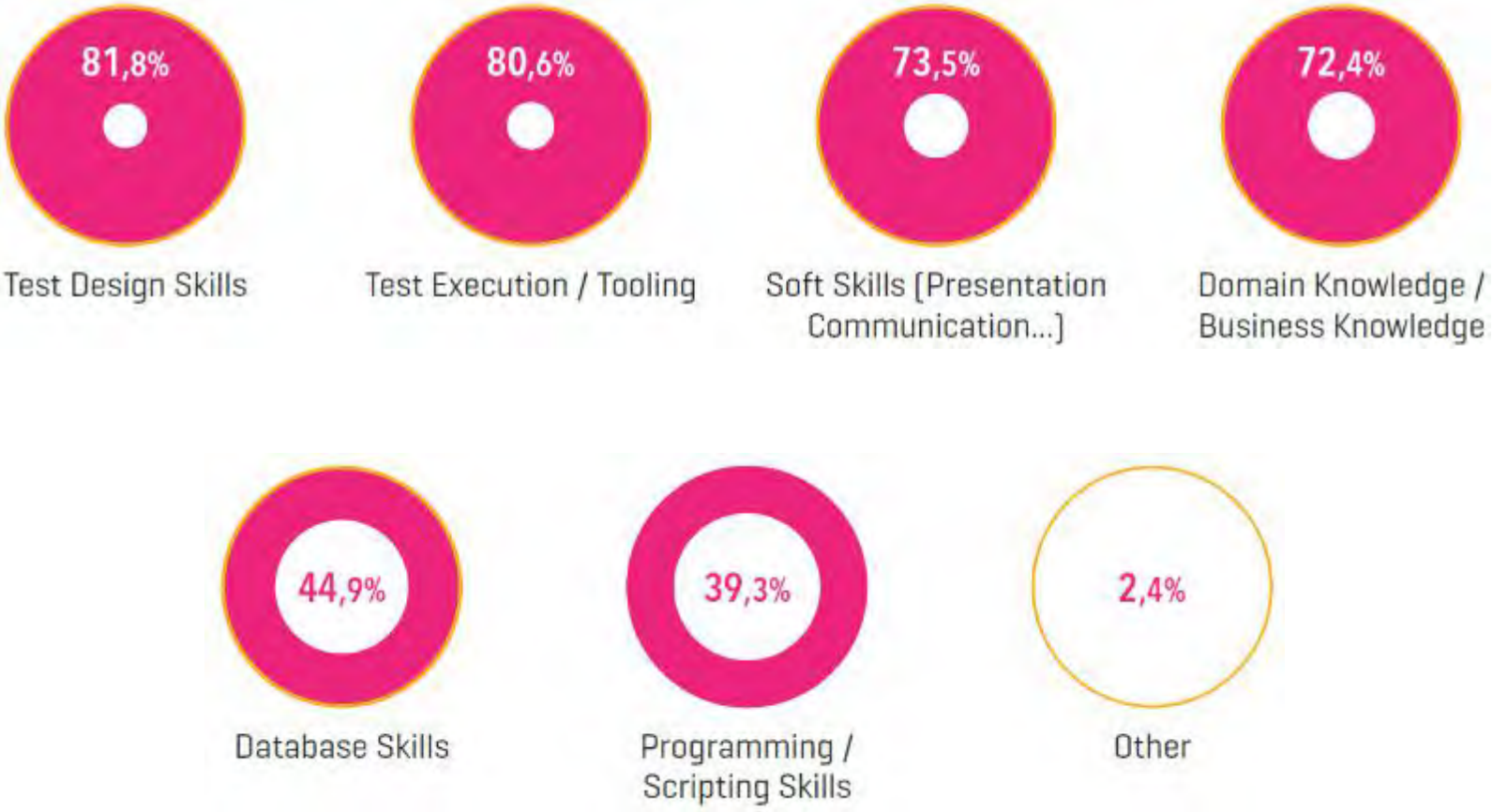
Team roles, examples

don't get locked up in a role

- Testing, test development
  - test analysis, test creation
  - reporting, result analysis and follow up, assessments
- Automation
  - functional navigation, technical automation, interface mapping
- Test execution planning and management
- Environments and infrastructure (环境 and 基础设施)
- Management
  - process, contents, practices, handling impediments
  - diplomacy (搞好关系)

testers often share the fate of defenders in football:
no glory, only blame

What companies expect



source: ISTQB 2015 - 2016

ABT in Agile

Agile life cycle

product owner

team

prod owner & team

Product Backlog ⇨ Sprint ⇨ Products

Test Module Definitions

Test Module Development

Interface Definition

Action Automation

Test Execution

Test re-use

Automation re-use

Test development

User stories
Documentation
Domain understanding

→

Acceptance Criteria
PO Questions
Situations
Relations

→ Main Level Test Modules

→ Interaction Test Modules

→ Cross over Test Modules

Using ABT in Sprints (1)

- Try to keep the testers in the same sprint as the rest of the team
  - don't let automated tests lack behind
  - note that in many environments tests and their automation are not highest priority (优先)
- Agree on the approach (同意作法)
  - is testability a requirement for the software?
  - do we regard tests, or some of the tests, as products?
    - can become part of the backlog
    - would not only be up to the team which tests to create
- Just like for development, use discussions with the team and product owners
  - deepen understanding, for the whole team
  - help define negative tests, and tests for unexpected situations (突发状况)
  - tests can work as driver for development (TDD and ATDD)

Using ABT in Sprints (2)

- Create good starting conditions for a sprint:
  - automation technology available (UI, non-UI, custom controls, graphics, ...)
  - know how you will deal with data and environments
  - understanding of subject matter, testing, automation, etc
- Do interface mapping by hand, using developer provided identifications
  - saves time by not having to use the viewer or other spy tools
  - recording of actions (not tests) will go better

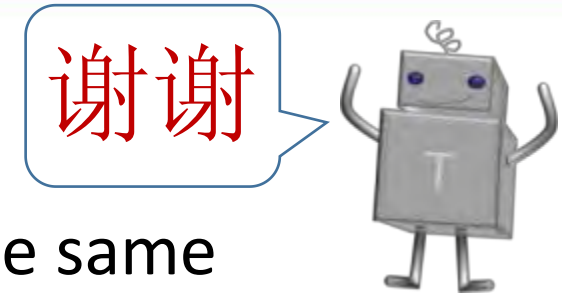
TIP

To discuss an approach, consider daily "sit down" meetings with some or all members to coach and evaluate

- an end-of-day counterpart to the early-morning "stand up" meetings
- short and friendly, not about progress and impediments, but about practices and experiences with them (like "what actions did you use?")
- a few meetings may suffice

Finally

Takeaways



- Not all "big project" challenges are the same
- Think before you do. Best results come from planning well, and combining effective concepts, tricks and tools
- Consider tests and automation as products that need design
- Team work is a key for short term and long term success
- There are many options for infrastructure, but keep an eye on economy, planning, and control