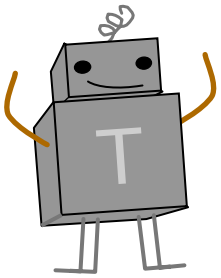
T3

Delivering Test Automation Success Through People, Methods & Tools



Hans Buwalda

LogiGear Corporation

hans @ logigear.com

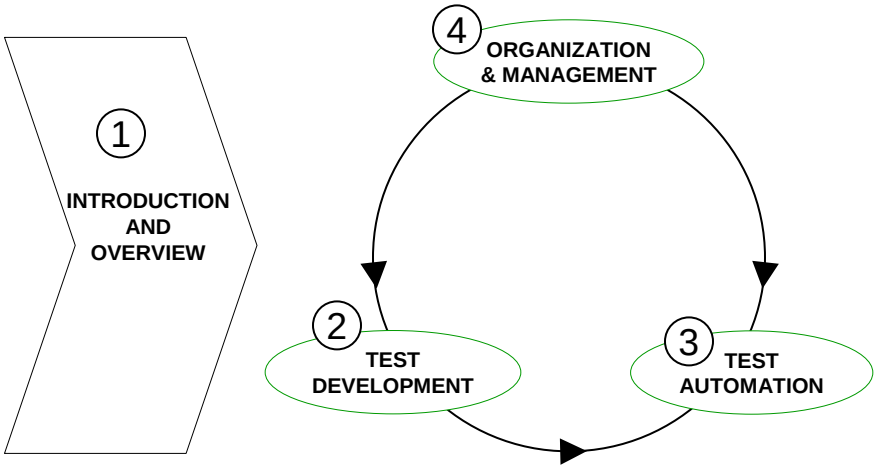
Introduction

- Who is here
- Which industries
- Affinity with testing
- Objectives for today

Objectives

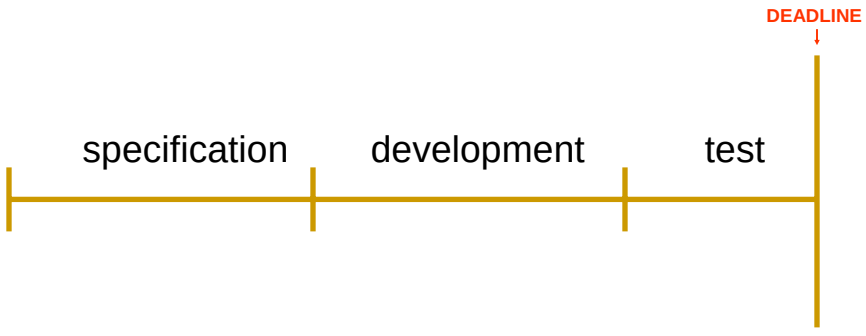
- Implement effective and maintainable automated tests
- Managing testing and test automation
- Main focus: concepts, techniques and experiences
  - To a lesser extend: in-depth treatments, full coverage of possible issues
- We will use the "Action Based Testing" method as a guideline today, as an example of an approach:
  - Representative of keyword-based approaches
  - Incorporates most of the key ideas and concepts
  - Home match for your teacher

Outline of this Tutorial

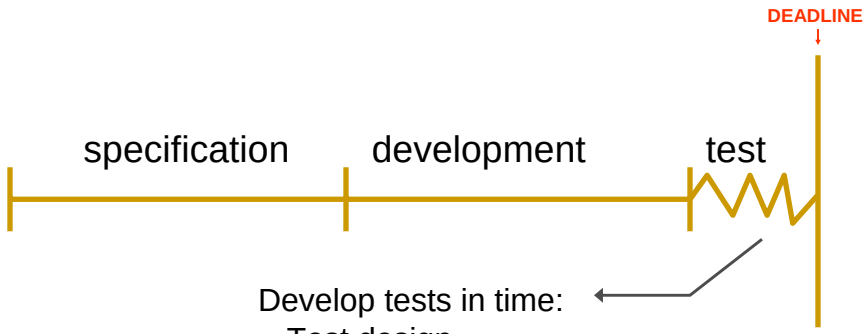


Introduction and Overview

Testing Under Pressure



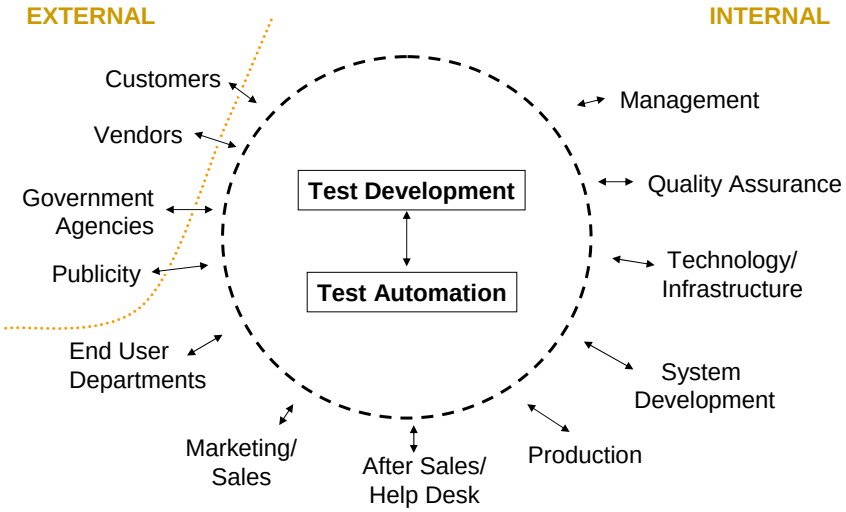
Testing Under Pressure



Develop tests in time:

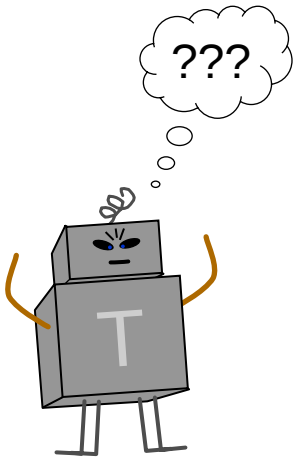
- Test design
- Auditing, acceptance
- Preparations
- Automation

Organize it Well



Existential Questions

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



Why test?

- People expect us to
- Somebody wants us to
- Increases certainty and control
  - Showing absence of problems
- Finds faults, saving time, money, damage
  - Showing presence of problems

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
- Reduces involvement of valuable specialists
  - "cans" application business and application knowledge
- Consolidates a structured way of working
  - when established as integral part of system development process
- Speeds up development
  - shorter turnaround between changes
  - more security

Why not Automate?

- Can rule out the human elements
  - promotes "mechanical" testing
  - might not find "unexpected" problems
- More sensitive to good practices
  - pitfalls are plentiful
- Creates more software to manage
- Needs/uses technical expertise in the test team
- Tends to dominate the testing process
  - at the cost of good test development

The Challenges for a Test Process

- Testing should be fun
- Testing should be effective
- Testing should be efficient
- Testing should be under control

Different Points of View

1. Does the system comply with the requirements
   → the “positive” angle

2. Are there any problems we should know about
   → the “negative” angle

3. Will the system work in practice
   → the “pragmatic” angle

Key Issues in Testing

- Appropriate test design
- Comprehensive automation architecture
  - manageable, maintainable
  - control over the technology
- Management of the tests
  - tests and test scripts are products that need to be managed
- Management of the test process
  - managers want to know what is going on
- Clear and useful reporting
  - progress, results
- Quality assurance
  - efficient and effective involvement of stake holders, users, auditors

What are Typical Problems

- No clear direction
- Test design is not well thought through
- Automation lacks architecture, is not transparent and hard to manage
- Testing is underestimated or avoided
  - testing is difficult and expensive
  - management is often in denial
  - it looks unattractive to spend money on testing
- Test automation is underestimated
  - test automation can be **very** difficult
- Focus on tools and technology, at the cost of test design

Success Factors for Automation

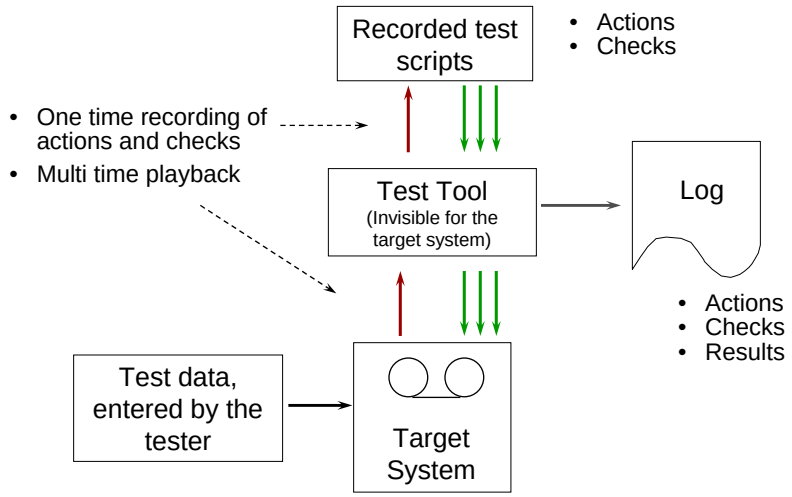
- Test design
  - more important than the automation technology
  - most underestimated part
- Comprehensive automation architecture
  - manageable, maintainable
  - control over the technology
- Organization and management
  - management of the tests
  - management of the test process
  - efficient and effective involvement of stake holders, users, auditors

Approaches to Automate Test Execution

- Record & Playback (or "Capture & Replay")
- Scripting (programming)
- Action Words (keywords)

see also: "Software Test Automation", Mark Fewster and Dorothy Graham,

Record and Playback



Record and Playback

```
select window "Logon"
enter text "username", "administrator"
enter text "password", "testonly"
push button "Ok"
select window "Main"
push button "New Customer"
expect window "Customer Information"
select field "First Name"
type "Paul"
select field "Last Name"
type "Jones"
select field "Address"
type "54321 Space Drive"
   .
   .
   .
```


Impressions of Record and Playback

- Can help to get to know a test tool, or the UI of the system under test
- Could work well for small tests
- For large tests, maintainability of the generated test scripts is a major risk factor
- Various test tools still seem to endorse it
  - “record and playback 2.0”

Scripting (programming the test cases)

- Test cases are designed first and then automated
  - Automation is managed as a programming task
  - Done by a specialized Automation Engineer
- Emphasis on structure and re-use
  - Similar to regular programming projects
  - Common tasks (like "logon") are factored out
  - Often data is separated from program logic, by moving it into separate data files ("data driven")
- Creation and maintenance like any other software
  - Superior to record&playback
  - But a lot of work, and a significant burden on engineering capacity
  - Very hard to manage
  - Also sensitive to changes in the application under test
  - Generally achieves a good quality, but a low automation coverage

Example Scripting

```
. . .
Function EnterCustomer(FirstName, LastName, Address)
        Click("New Customer");
        ExpectWindow("New Customer");
        EnterField("First Name", FirstName);
        EnterField("Last Name", LastName);
        EnterField("Address", Address);
        . . .
        Click("OK");
End Function

. . .

Function Main()
        Logon();
        EnterCustomer("Mary", "Jones", "123 Palm Drive");
        EnterCustomer("Paul", "Franklin", "321 Regent Street");
        . . .
        LogOff();
End Function
. . .
```

Impressions of Scripting

- Valuable approach
- Can be modified to work “data driven”
- Well-supported in most test tools
  - many useful functions available
  - even the built-in scripting languages allow for decent automation development
- It focuses much on automation design
  - can go at the cost of test design
  - if all scripts are programmed, it leads to a lot of code to manage

Key Success Factors for Automation

- Accessible and maintainable test structure
- High degree of automation
- High re-use
- At least as maintainable as the system under test
- Test specification separate from automation
  - At the same time avoiding double work between tester and engineer

The 5% Challenge for Test Automation

- No more than 5% of all efforts around testing should involve automating the tests
- No more than 5% of all tests should be executed manually

Action Words / Action Based Testing (ABT)

- Provides a framework for effective automation
- Based on a notion that a test can be broken down in a number of consecutive actions
- Both actions and their data are in products, called "Test Modules"
  - Excel spreadsheets for easy development and communication
  - test data is explicit or with a place holder
  - explicit checks with specified expected result values
  - most actions are "high level", omitting as many unneeded details as possible
- Instead of implementing test cases, the automation engineer concentrates on the programming individual actions

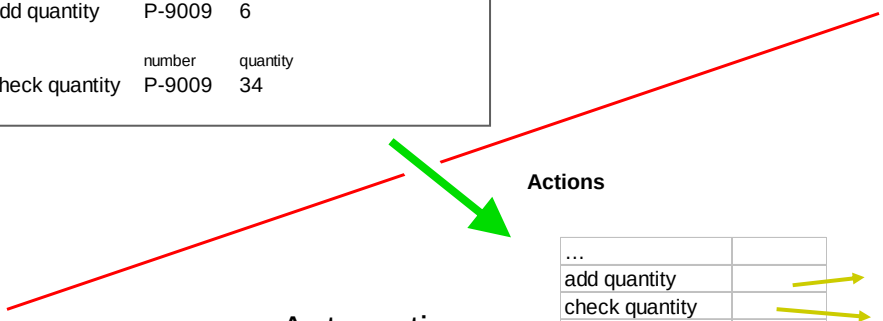
Example of a Test Module

TEST MODULE	**Product Maintenance**		
VERSION	1.0		
. . .			
TEST CASE	**Test Case 03**	**Use an invalid number**	
test requirement	add 003	A number must be divisable by 9	
	source	control	destination
click	main	new	new product
	window	control	value
enter	new product	number	P9001
	source	control	
click	new product	process	
	text		
check message	Invalid product number.		

Separation of Tests and Automation

	number	name	quantity
new product	P-9009	Sledge Hammer	5
	number	quantity	
add quantity	P-9009	20	
add quantity	P-9009	3	
add quantity	P-9009	6	
	number	quantity	
check quantity	P-9009	34	

Test Module
(Excel Spreadsheet)



Example of "Old and New" Formats

Most values are implicit. The tester has to figure them out during execution….

Same instruction is repeated over and over again…

Current format

Enter a user id that is greater than 10 characters, enter proper information for all other fields, and click on the "Continue" button	An error message should be displayed stating that "User Id must be less than 10 characters".
Enter a User Id with special character's), enter proper information for all other fields and click on the "Continue" button	An error message should be displayed indicating that "User Id cannot contain some special characters".
Enter the information, with a password of 4 characters and click on the "Continue" button	An error message should be displayed with "Password must contain at least 5 characters".

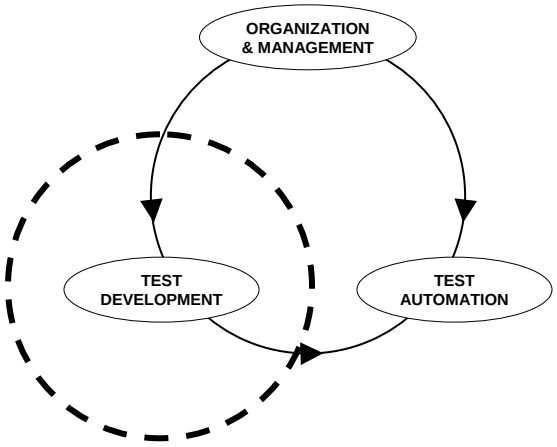
ABT format

	user id	message
check registration dialog	aaaaabbbbbc	User Id must be less than 10 characters
	user id	message
check registration dialog	résoudre	User Id cannot contain some special characters
	password	message
check registration dialog	test	Password must contain at least 5 characters

Impressions of Action Based Testing

- Strong emphasis on test design
  - but can easily be misused
- Good advantages, in particular for larger test sets
  - best possible maintainability
  - tests can be readable and easily understandable
- Technically a simple model
  - most test tools have the necessary functionalities in one form or another
- It is an involved method
  - needs to be well-organized, including proper design, automation build-up and training of all involved
  - less suitable for small ad-hoc automation jobs
  - easy to do it “wrong”

Test Development



Coverage, the "MacGuffin" of Testing

- The thing everybody is looking for
  - movie term, introduced by Hitchcock
  - the characters consider it extremely important, but it is not necessarily known what it is exactly
- Many forms of coverage
  - Technical
  - Functional
  - Business wise
  - Risk based
  - Requirement based
- Own experience: a large portion of bugs are not straightforward
  - not tight to one particular code line, function or requirement

Explicit Test Requirements

...
TR-3.51 The exit date must be after the entry date
...

```
// Test Case: Billy Goodfellow
// Test Requirement: TR-3.51
Function EnterBilly
        Press "New Employee"
        Enter Field "Name", "Goodfellow"
        Enter Field "First Name", "Billy"
        Enter Field "Entry Date", "2002-10-02"
        Enter Field "Exit Date", "2002-10-01"
        . . .
        Press "Process"
        Find Text "Exit date must be after entry date"
        if Found then
           Result passed
        else
           Result failed, "No error message"
        . . .
End Function
```


Explicit Test Requirements

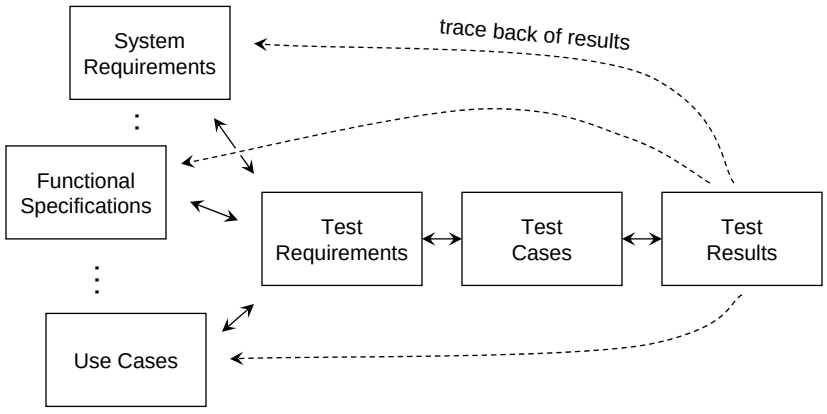
...
TR-3.51 The exit date must be after the entry date
...

test requirement	TR-3.51		
	name	entry date	exit date
enter employment	Bill Goodfellow	2002-10-02	2002-10-01
check error message	The exit date must be after the entry date.		

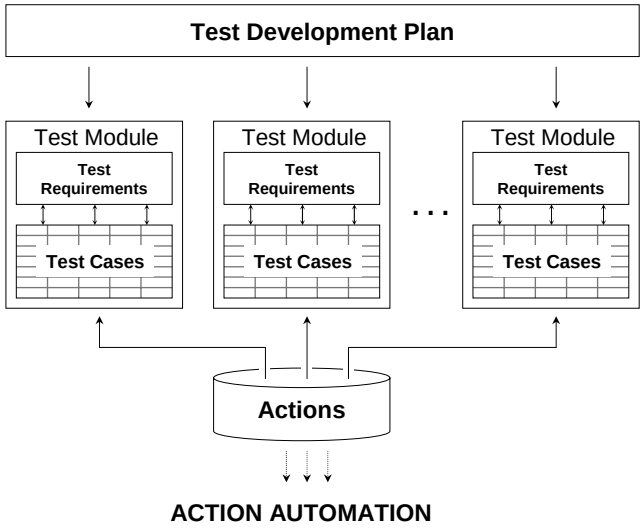
Matching Conditions

Test Req	Description	Severity	Tested in Test Case:	
Cust-TR01	A customer entered in the enter client screen is present in the database	high	Cust-TC01	Entering customers using manual numbering
			Cust-TC02	Entering with system generated numbering
Cust-TR02	A customer with a positive balance can transfer money to another customer	high	Cust-TC01	Entering customers using manual numbering
			Cust-TC02	Entering with system generated numbering
Cust-TR03	**A customer with a negative balance cannot transfer money to another customer**			
Cust-TR04	The balance of the paying customer is decreased with the sum payed	high	Cust-TC01	Entering customers using manual numbering
			Cust-TC02	Entering with system generated numbering
Cust-TR05	The balance of the receiving customer is increased with the sum payed	high	Cust-TC01	Entering customers using manual numbering
			Cust-TC02	Entering with system generated numbering
Cust-TR06	Account numbers can be entered manually by the user	medium	Cust-TC01	Entering customers using manual numbering
Cust-TR07	Account numbers can be generated automatically by the system	medium	Cust-TC02	Entering with system generated numbering
Cust-TR08	...			
...				

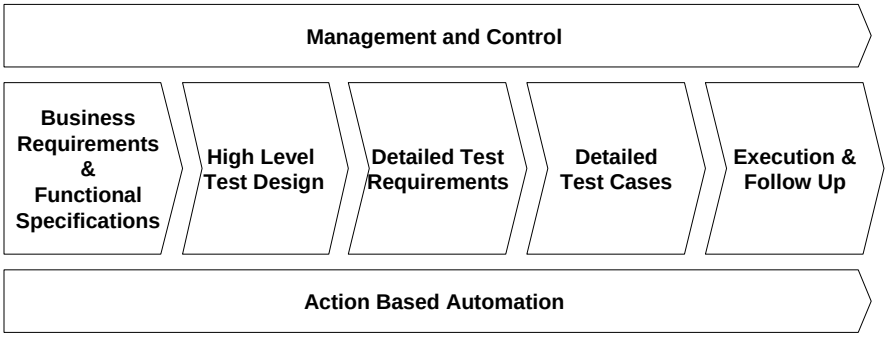
Relation to Source Information



Test Development Products

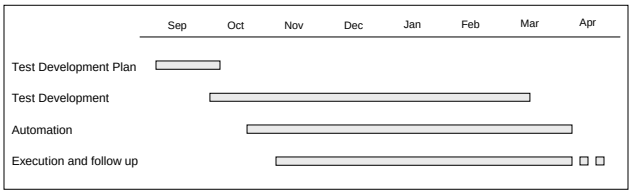


Typical Life Cycle

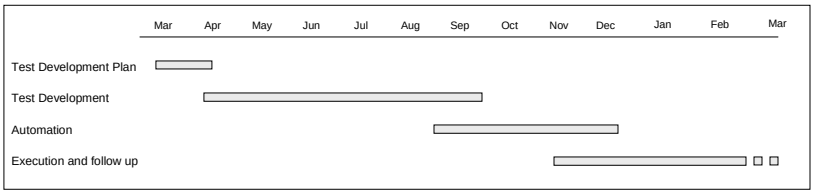


Examples of Time Lines

System under test becomes available in phases



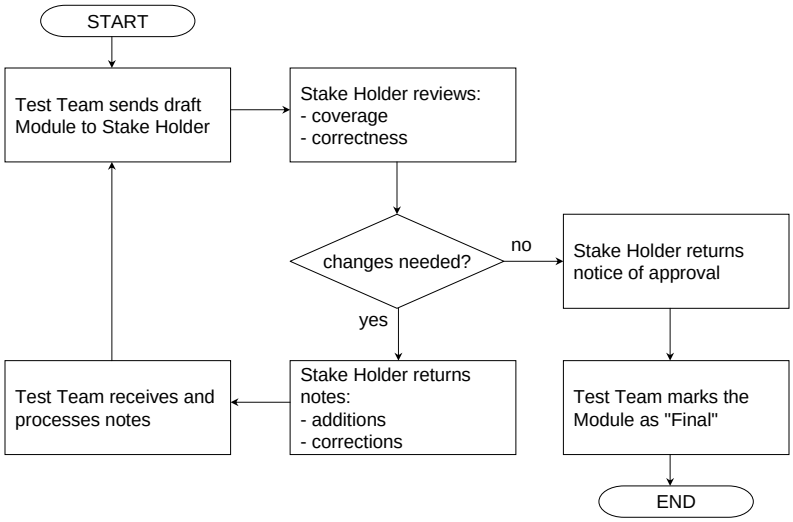
System under test becomes end of the cycle



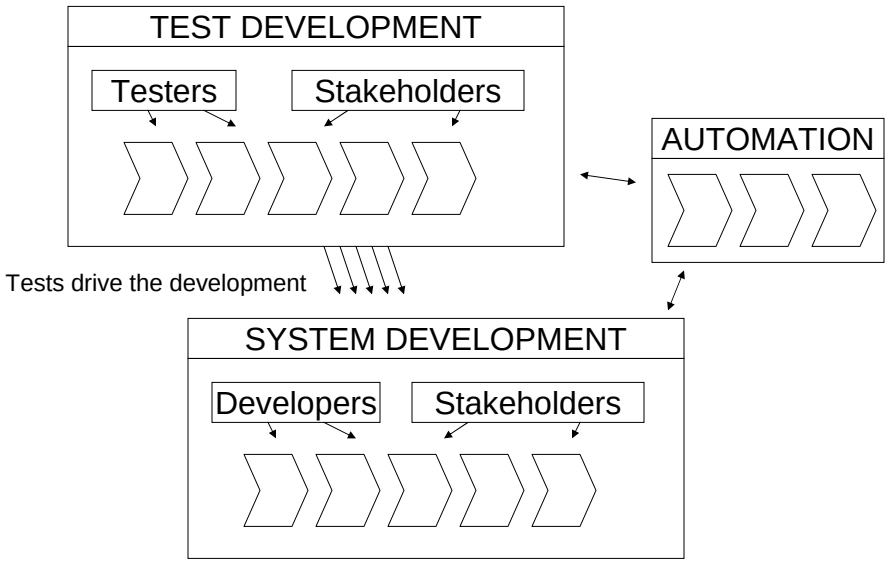
Example Module Development Planning

Nr	Module	Business Owner	Date to BO
1	Portal Navigation, Audience	Robyn Peterson	05 / 23
2	Portal Navigation, Search	Ted Jones	05 / 27
3	Membership, registration	Steve Shao	06 / 03
4	Portal Navigation, Category	Ted Jones	06 / 08
5	Portal Navigation, Topic and Expert	Ted Jones	06 / 13
6	Access Control	Mike Soderfeldt	06 / 17
7	Portal Navigation, Task	Ted Jones	06 / 22
8	Contact DSPP	Ted Jones	06 / 27
9	Portal search	Mike Soderfeldt	07 / 01
10	Membership, review and update	Steve Shao	07 / 05
11	Program contact assignment	Alan Lai	07 / 11
12	Company, registration	Steve Shao	07 / 14
13	Catalog, view and query	Robyn Peterson	07 / 19
14	Site map	Ted Jones	07 / 25
15	Membership, affiliation	Steve Shao	07 / 28
16	Learn about DSPP	Ted Jones	08 / 01
17	Products and services	Steve Shao, Robyn Peterson	08 / 08
18	What's new	Ted Jones	08 / 11
19	Company, life cycle	Steve Shao, Alan Lai	08 / 17
20	Specialized programs	Ted Jones, Steve Shao	08 / 22
21	Customer surveys	Ted Jones	08 / 29
22	Software downloads	Mike Soderfeldt	09 / 01
23	Newsletters	Ted Jones	09 / 06
24	Internationalization and localization	Ted Jones	09 / 13
25	Membership, life cycles	Steve Shao	09 / 19
26	Collaboration, forums	Ted Jones	09 / 23
27	Collaboration, blogs	Mike Soderfeldt	09 / 28
28	Collaboration, mailing lists	Ted Jones	10 / 03

Review Process with Stake Holders



Agile ABT



The Three “Holy Grails” of Test Design

Establishing the modules

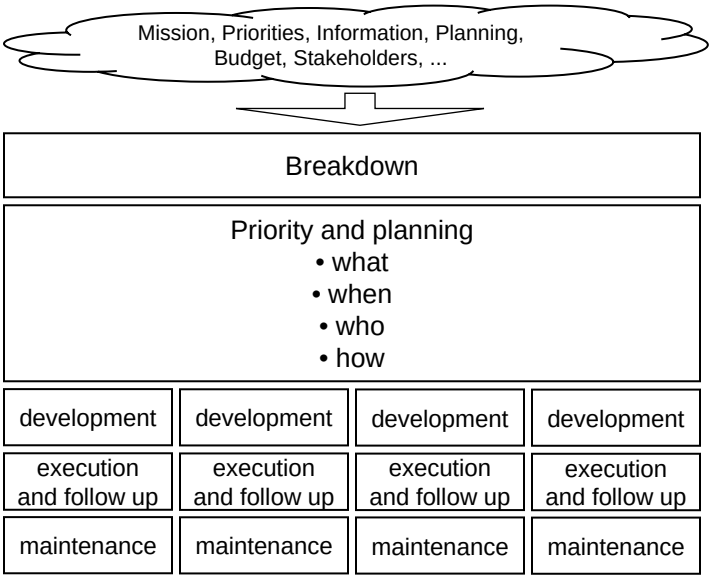
The right approach per module

The proper level of test specification

The First Grail

Establishing the modules

Divide and Conquer



Symptoms of a Failing Test Design

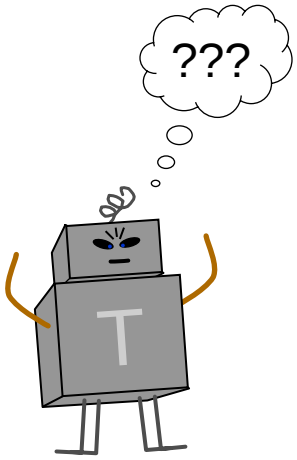
- Combinations of low and high level tests in one test module
  - like functional and UI checks in one test case
- When you have to change all test modules for a new system release
  - most test modules should not be sensitive to system under test changes
- If all test modules look alike
  - all have the same kind of tests
- Test modules are dependent on each other
  - when you want to run module x, you need to run module y first
  - this will make test runs longer and more tedious
- You can't start developing them early in the life-cycle
  - all need detailed system information, there are no high-level test modules
- It is hard to get the automation to work
  - and to keep it working across system changes

Properties of a good Breakdown

- Logical to all concerned
- Independent from other modules
- Well differentiated and clear in scope
- Fitting the priorities and planning of the project
- Balanced in size and amount

Question

- Can you give a set of breakdown criteria for tests?



Breakdown "Cheat Sheet"

- Straightforward Criteria
  - Architecture of the system under test (which part are we testing)
  - Functionality and other requirements
  - Kind of test (navigation flow, quality attributes, ...)
  - Ambition level (smoke test, regression, requirement based, aggressive, …)
- Additional Criteria
  - Stakeholders
  - Complexity of the test
  - Technical aspects of execution
  - Planning and control
  - Risks involved

Splitting into Test Modules

- UI oriented tests
  - does function key F4 work
  - does listbox xyz the right values
  - is the tab order correct
- Do individual functions work
  - like transactions in a financial system
- Alternate paths in use cases
  - like cancel a transaction
- End-to-end tests
- Simulating business processes
- Tests with specific automation needs
  - like multi station tests
- Tests of non-UI functions
- High ambition level tests (aggressive tests)

The Second Grail

Approach Per Module

Eye on the ball, Scope

- Always know the scope of the test module
- The scope should be unambiguous
- The scope determines many things:
  - what the test requirements 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)

Examples of Testing Techniques (1)

- Equivalence class partitioning
  - any age between 18 and 65
- Boundary condition analysis
  - try 17, 18, 19 and 64, 65, 66
- Error guessing
  - try Cécile Schäfer and test the sorting of the name list
- DAST
  - Diagnostic Approach to Software Testing
  - works by asking systematic questions to get to test cases
- Exploratory
  - "Exploratory testing is simultaneous learning, test design, and test execution", James Bach, www.satisfice.com
- Model-based testing
  - see Harry Robinson: www.model-based-testing.org
  - see STQE Magazine, March 2003
- Error seeding
  - deliberately injecting faults in a test version of the system, to see if the tests catch them
  - handle with care, don't let the bugs get into the production version

Examples of Testing Techniques (2)

- State transitions
- Decision tables
- Soap Opera Testing

Other sources: "Why Software Fails", James Whittaker and Alan Jorgensen, and "Testing Computer Software", Cem Kaner, Jack Falk, Hung Nguyen

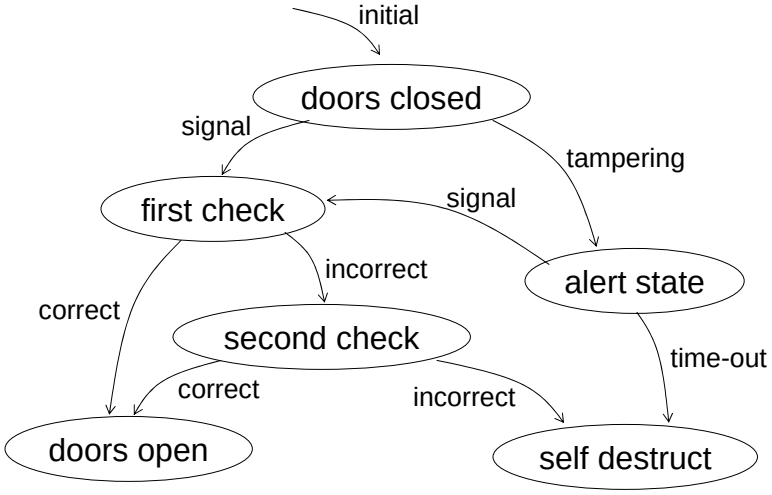
State Transition Table

Car Safety System, Model 007, Luxury Version*

NR	STATE	EVENT	OUTPUT	NEXT
1	doors closed	signal	beep	first check
2	doors closed	tampering	hi-lo	alert state
3	first check	correct code	green	doors open
4	first check	incorrect code	hi-lo	second check
5	doors open	close signal	beep	doors closed
6	second check	correct code	green	doors open
7	second check	incorrect code	hi tone	self destruct
8	self destruct	anything	boom	oblivion
9	alert state	time out	hi tone	self destruct
10	alert state	signal	beep	first check

*certain insurance restrictions might apply, please check with your agent

State Transition Diagram



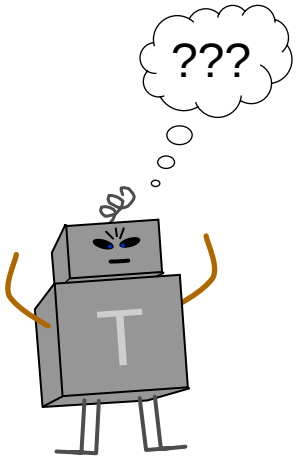
Example of a Decision Table

dialog data correct	n	y	y	y	y
large order		n	n	y	y
online customer		n	y	n	y
show error and resume	x				
give discount 20%				x	x
give discount 3%			x		
notify marketing				x	x
send to priority execution			x		
display price info		x	x	x	x

Case 1
Case 2
Case 3
Case 4
Case 5

Question

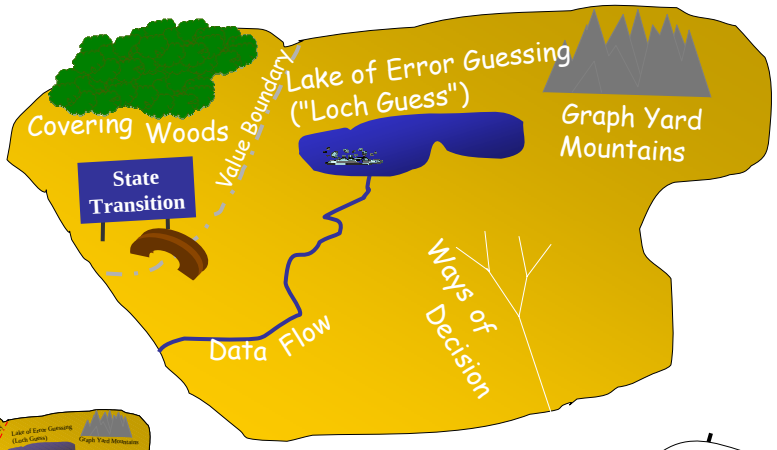
- What is wrong with the following pictures?

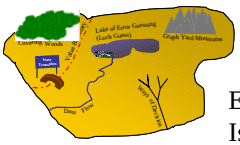


No Y2K Problems in Auckland Airport??

Should we Stop?

The Realm Of Techniques





Equivalence
Island

The "Mechanical Approach" for Test Development

- Start with a (preferably long) list of requirements
- Make a test case for every requirement
- Use a standardized test technique to translate the requirements into test cases
- Hire (many) people to perform the tests manually
- ….

Some Pitfalls With a Too Mechanical Approach

- No fun at all
- Inhibiting creativity
- Coverage is focused at single requirement level
- Any defects should probably have been found in an earlier test
- Suggests false sense of control
- Test set hard to maintain
- Doesn’t catch mistakes in the requirements

Soap Operas

Ashley hears about Jack's deposit when he thought he had to go. Victoria lectures her father about what's wrong with him and Nikki but Victor advises her that it's none of her business Olivia learns Dru has no regrets about leaving and takes great satisfaction in having Lily as her companion. Dru then asks Olivia why she is raking Malcolm over the coals. Stopping by Gina's, Nikki spots Brad and sits with him, admitting she doesn't want to be alone tonight. Victor stops by Mack's party at the Crimson Lights. Ashley takes a home pregnancy test. Worried about Billy, Raul makes call and J.T. claims he doesn't know where Billy is. Raul rushes over and finds Billy out cold in the snow. Raul worries when he can't find a pulse. . . .

Some Other Examples

Pension Fund

William starts as a metal worker for Industrial Enthropy Incorporated in 1955. During his career he becomes ill, works part time, marries, divorces, marries again, gets 3 children, one of which dies, then his wife dies and he marries again and gets 2 more children….

World Wide Transaction System for an International Bank

A fish trade company in Japan makes a payment to a vendor on Iceland. It should have been a payment in Icelandic Kronur, but it was done in Yen instead. The error is discovered after 9 days and the payment is revised and corrected, however, the interest calculation (value dating)…

Properties of Soap Operas

- About “real life”
- But condensed
- And more extreme
- Using a recurring theme, with “episodes”

Soap Operas for Testing

- Define a scope of the test to develop
- Identify with the business environment
- Identify which elements would make things difficult
- Draft test cases (typical some dozen lines)
- Write them down in modules

Example of test lines

	from	to	amount	valuta	trans nr
enter payment	123421344	4124244123	120000	yen	>> trans1
	trans id	amount			
check value dating	# trans1	$0.47			
	days				
simulate wait	9				
	trans id				
order to reverse	# trans1				
	from	to	amount	valuta	trans nr
enter payment	123421344	4124244123	1200000000	IKr	>> trans2
	trans id	amount			
check value dating	# trans2	$7,701.56			

. . . .

Soap “Count the Goodies”

In this episode, the Goodys have won $750,000 in the lottery. They decide to buy that big house on the corner of the street that they’ve always dreamed about. Father, Bing Goody, goes to the banking office to make the necessary arrangements. Of course, the bankruptcy that Bing had two years ago complicates the approval, but Jim, the guy at the bank, is very helpful and even sells Bing on the idea of a beautiful vacation property in Mexico. Bing has a brilliant idea. Why not ask neighbor Jones if he is interested in co-owning the vacation property? Like everything else with Mr. Jones, his credit is perfect.

The Story Continues…

One week later: After a small party with some new friends, only about $720,000 of the prize money is left, so the mortgage arrangements need to be changed. Jim, Mr. Jones, and Bing decide to meet the next morning to make the necessary adjustments. Since it was a great party, they are not at their best when they meet, so many mistakes are made. They correct some immediately, others they miss and will need to fix later. They find that an additional second mortgage on Mr. Jones’s home is needed in order to still qualify for the vacation home.

What Will Happen Next?

See it all . . .

. . . in the next module!!

What This Soap Can Test

- Entering a customer with a big family
- Mortgage arrangements including a first and a second house
- Property abroad
- Both a primary residence and a vacation property
- Weighing of the income of a second owner
- Co-ownership on the second house only
- Establishing a second mortgage to finance a vacation property
- Bankruptcy two years ago
- Modification of the down payment
- Correcting errors upon entry
- Corrections after the application has been processed

Soap Operas (in Testing) are not Necessarily

- "Extreme"
- Far fetched
- Long and elaborate
- Pieces of art and creativity

"Killer Soaps"

- More specifically aimed at finding hidden problems
- Run when everything else has passed
- One option: put a killer soap at the end of a normal module
- Ask the "specialists" for input

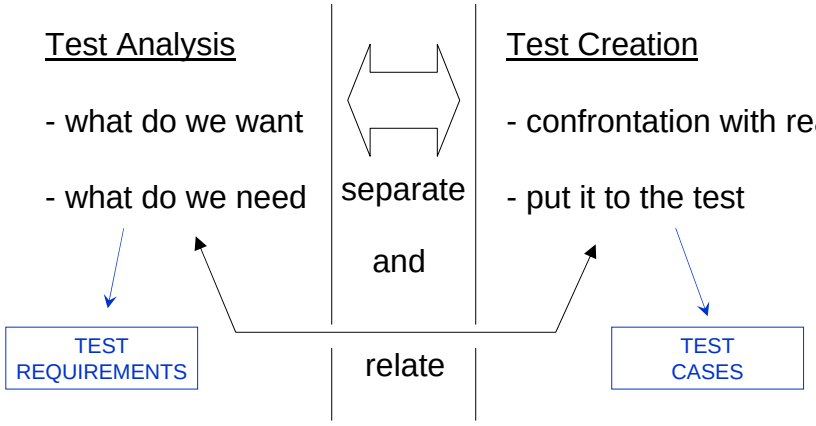
Making Test Requirements

- Test requirements are an "analysis" product (in contrast to test cases, which are "design" products). Make sure you understand functionality well before writing the test requirements
  - Avoid "copy and paste" of literal text from specifications and requirements
- Avoid "copy and paste" text from specifications or requirements
  - The reader should
- Make cause and effect clear, and mention cause first ("clicking submit empties all fields")
- Make condition and effect clear, and mention condition first ("if all fields are populated, ok is enabled")
- Split complex sentences into small, atomic, statements
  - It is ok tough to combine two or more functionalities if this is not adding to complexity (like "ok becomes enabled if both first name and last name are specified")

Making Test Requirements

- Keep test requirements short. Leave out as many words as you can without loosing the essential meaning
- Try to keep test requirements independent from each other
  - Exceptions are possible though. For example a specific situation in the system under test might need a group of test requirements to describe it. In such a case try to use something like
    - "in the case of a single cell: ...."
    - "in the case of a single cell: ...."
    - "in the case of a single cell: ...."
    - ....
- Within a test module, group the test requirements as logically as possible
- Based on the organized and well-formulated test requirements, try to identify gaps and either add test requirements if they are obvious, or provide feedback to the customer in case of questions

Test Analysis / Test Creation

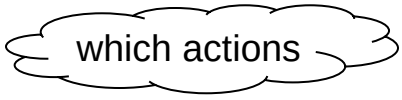


Questions to Answer with a Test Collection

		mechanical	soaps	soaps + techniques
1	Does the system comply with the requirements	***	*	**
2	Are their any problems (defects and/or failures) we should know about	*	***	***
3	Will the system work in practice	**	**	***

The Third Grail

Level of Test Specification



Probably not a Good Design

- Too high level to understand the test:“Test all”

```
start program
test everything
end program
```

- Low-level actions (automation) in a high-level test:

```
type            "Hans"
press key       "<tab>"
type            "Buwalda"
click button    "OK"
```

- Hard to understand “insiders only” language

```
set code    Fc122x   XX33
```

- Tons of arguments not relevant to the test
  - 85 arguments for an action
  - have sensible default values for common arguments like zip codes, phone numbers etc

Typical Consequences / Symptoms

- Tests become quite unreadable (especially for non experts)
- Unpleasant work to make the tests
- Hard to understand the results
- High risk of mistakes
- And, of course, heavy maintenance dependency

Designing Actions/Specification Level

- Scope of the test determines the specification level
- As high level as appropriate, as little arguments as possible
  - Use default values for non-relevant arguments
- Clear names (usually verb + object works well)
- Manage the Actions
- Document the Actions

Identification of Actions

- Specification of an action, a check or a documentary statement
- Communication between Automation and Test Cases
- Consistent
- Standard
- By-product of the test analysis

Maintenance of Actions

- Actions are a test product
- However the navigator is the best person to manage them
- Keep them in a registry
  - database
  - text document
  - spreadsheet

Levels of Actions

- High level actions to hide details
  - to hide unneeded details
    - always hunt down details
  - implemented in terms of lower level actions
- Low level actions to access the automation technology
  - either built-in in TestArchitect
  - or create yourself in a “harness”
    - this is the only real programming you would do

Example of “Going High Level”

	window	tree	tree item path
click tree item	main	my project	# "projects/" & project1
	window	max	
wait for window	main	# shortwait	
	window	list	item
check item exists	main	case list	Master Account

	project	item
check project entry	# project1	Master Account

Example of “Going High Level”

section	Get cost values			
	key			
key navigate	F7			
key navigate	5			
	window			
wait for controls loaded	view account info			
	text			
check breadcrumb	# view account info breadcrumb			
	window	field		
check focus	view account info	cancel		
	start cost	cost in	cost out	net cost
capture cost values	>> starting cost	>> total cost in	>> total cost out	>> total net cost

Example of “Going High Level”

section	Get status			
	window	menu	item	
click select menu	general look	menu bar	system	
	keys			
type	{S}			
	window			
check window exists	status information			
	cost no	logon	dep level	cost min limit
capture status information	>> dep num	>> logon_id	>> account level	>> cost min
	cost max limit	shop	region	branch
>>>	>> cost max	>> shop	>> region	>> branch
	source	control		
click	status information	ok		

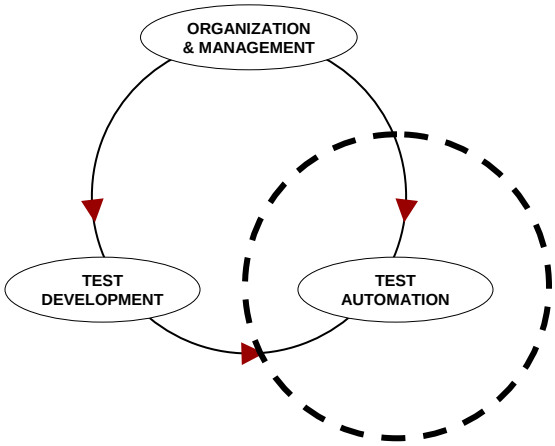
Example of “Going High Level”

section	**Capture initial number**		
	key		
key navigate	F7		
key navigate	3		
	page tab		
locate page tab	Scan Criteria		
	window		
wait for controls loaded	search		
	text		
check breadcrumb	# search breadcrumb		
	window	control	value
select	search	scan direction	Backward
	window	control	value
enter value	search	business date match	# bus date
	source	control	
click	search	go	
	window		
wait for controls loaded	search results		
	store as		
capture sequence num	>> seq num		

Example of “Going High Level”

	start cost	cost in	cost out	net cost
get cost values	>> starting cost	>> total cost in	>> total cost out	>> total net cost
	cost no	logon	dep level	
get status info	>> dep num	>> logon_id	>> account level	
	shop	region	branch	
>>>	>> shop	>> region	>> branch	
	cost min limit	cost max limit		
>>>	>> cost min	>> cost max		
	number			
get seq number	>> seq num			

Test Automation



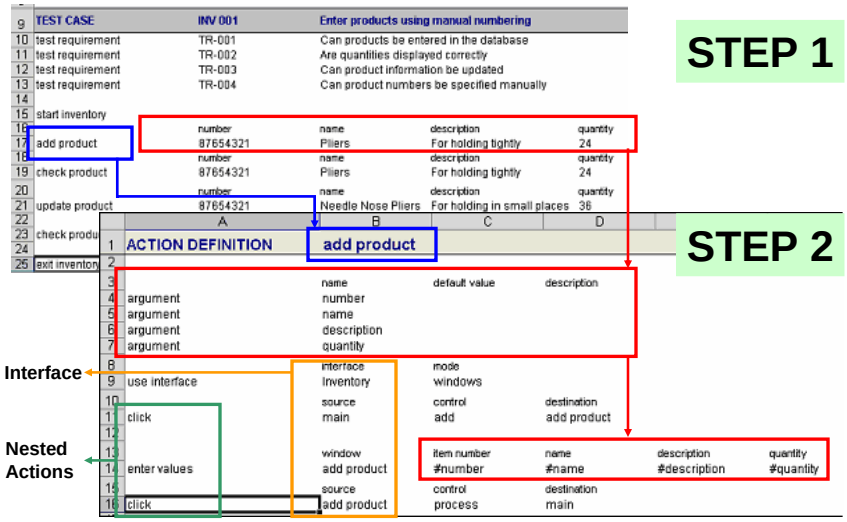
Components of Automation

- Tests and test data
  - what needs to happen to prove or disprove the functionality under test
  - best to keep this separate from the technical part of the automation
- Programming/scripting
  - technical specification of the individual steps of the automation
- Interface information
  - what is where on screens, web pages, etc
  - shielding volatile “physical” details
- UI and non-UI automation functions
  - the actual (usually tool provided) commands to work with the interfaces of the system under test

Implementing Actions

- In a keyword driven approach like Action Based Testing, the automation focuses on the actions
  - as opposed to programming test cases
  - in general this will lead to fewer and shorter scripts etc in the automation
- In the ABT method actions are implemented in three ways:
  - built-in in the tool, like “click”, “enter”, “check” etc
  - defined in an “action definition”, expressed with other (“low level”) actions
    - similar to functions in a programming language
  - programmed in a programming or scripting language
    - using what I like to call a “harness”

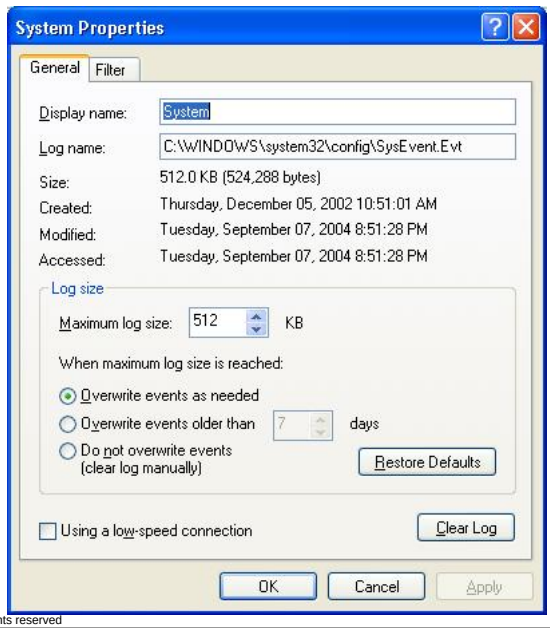
Action Definitions



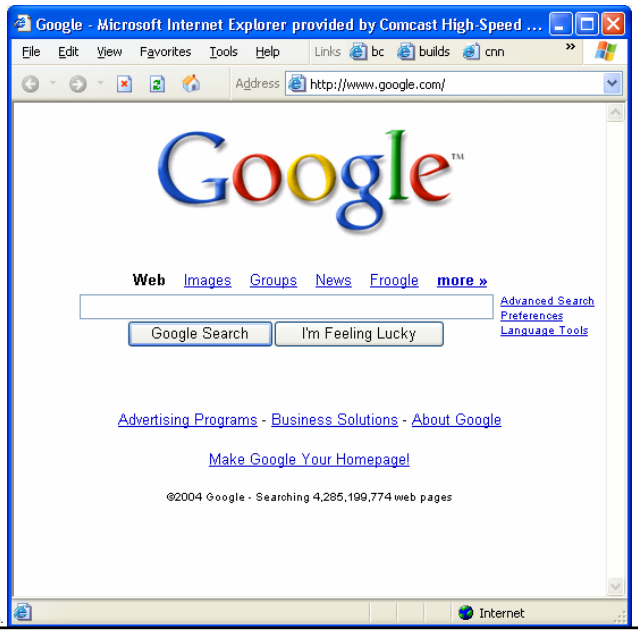
Interface Information and Interface Handling

- Handling interfaces is the essential challenge
  - automation works "outside in"
  - interfaces are not necessarily designed with test tools in mind
- Common interfaces
  - GUI
  - Web
  - API's, with methods and/or functions
  - message protocols, on basis of TCP/IP or SS7
  - embedded software
  - databases with SQL
  - files (like in batch runs)

Example of a GUI



Example of a Web Interface



Identifying Elements of an Interface

- All elements have properties
  - can be used for identification and verification
- Some elements can be easily identified
  - like buttons, checkboxes, frames
- Other elements have only volatile and/or ambiguous properties
  - like text boxes, list boxes, combo boxes
  - can differ across runs, or across versions/builds
- Text labels tend to be "invisible"
  - only bitmap

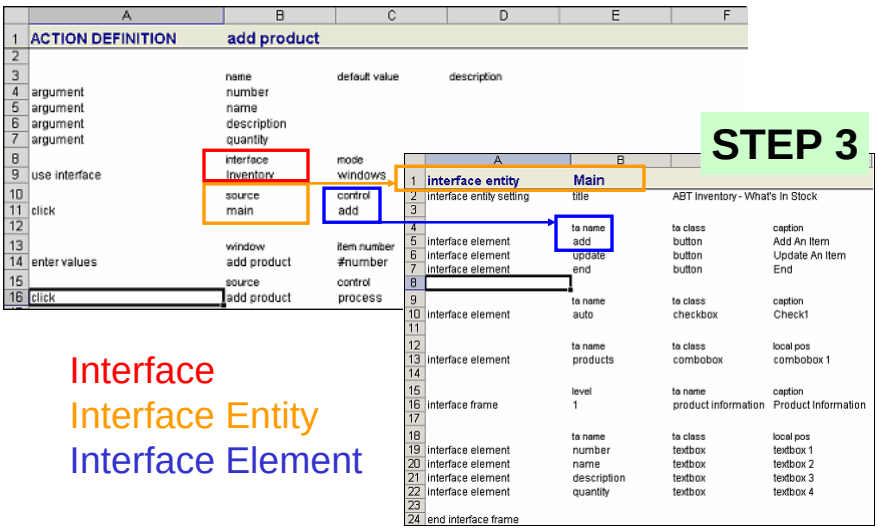
Interface Definitions

- Quickly define how interfaces must be accessed
- Commonly used for GUI's, however applicable to other interfaces as well
- Typically kept in a table or spreadsheet
  - separate from the test modules

Components of Interface Definitions

- Interface entity
  - window
  - web page
  - text screen
  - 3270 page (typically based on the HLLAPI standard)
  - class member
  - api function
  - message
  - database table
- Interface element
  - button
  - text box
  - link
  - field
  - argument
  - ....

Interface Definition



Interface
Interface Entity
Interface Element

Programming

- Often called "scripting", but much like regular programming
- Traditional playback tools mostly have built-in scripting languages
  - often Basic look-a-likes, WinRunner's TSL is a C look-a-like
- More and more tools support existing programming languages
  - like Visual Basic
- Regular languages can in some cases do the job too
  - in particular for non-gui's
  - for example scripting languages like Python and Ruby
  - but also Java, C++, VB, C#, etc

Functionality of an Action Interpreter

Function Interpret

While not end of file do

Read next line

Split the line into arguments

Look up the action in the "action list"

Execute the action

Print the results of this line in the report

End of the while loop

End

"Atomic and Generic"

- In ABT key recommendations for programmed actions are to make them:
  - atomic, only one small task per action
  - generic, maximum re-use
- Avoid programming test cases or high level actions
  - split them up in well-defined low level actions

Example: Performing a Check

- Check product p2 ... AAX

Product Information Screen

Product code	Product	Colour	Product Type	Weight
P1	screw	black	Orion	1
P2	nail	black	AAX	1

F1 Help F2 Select F9 Main Menu

module	**Test cases C1 Product**				
version	0.1				
date	05/01/1997				
author	T Ester				
test case	**C1L2**	**Product codes can be entered, if unique**			
	product code	product	color	type	weight
enter product	p2	nail	black	AAX	1
expect message	transaction processed				
	product code	product	color	type	weight
enter product	p2	nut	gray	AAX	1
expect message	the value in the field is not allowed				
	a product with a different code is allowed				
	product code	product	color	type	weight
enter product	p3	nail	black	AAX	1
	check if the product is realy present in the database				
	product code	product	color	type	weight
check product	p3	nail	black	AAX	1
delete button					

argument(0) argument(1) argument(2) argument(3) argument(4) argument(5)

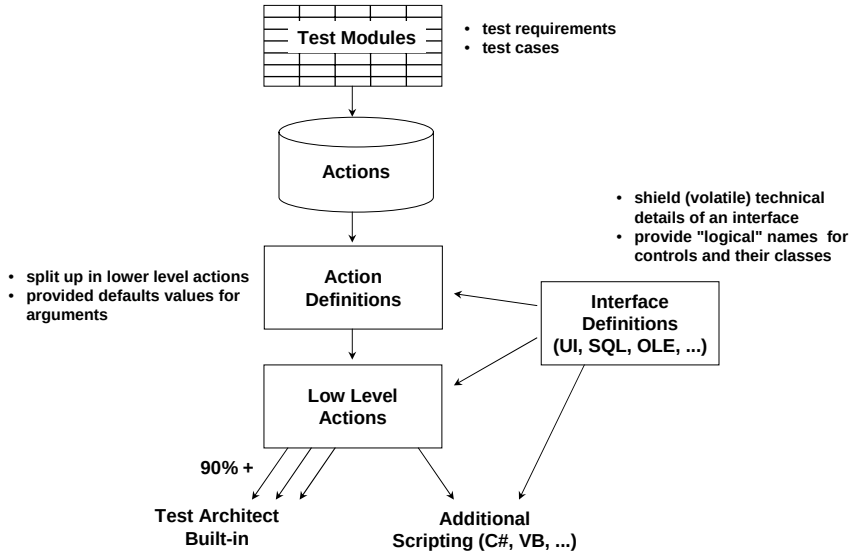
Scripting Example (Single Level Automation)

- Check product p2 ... AAX

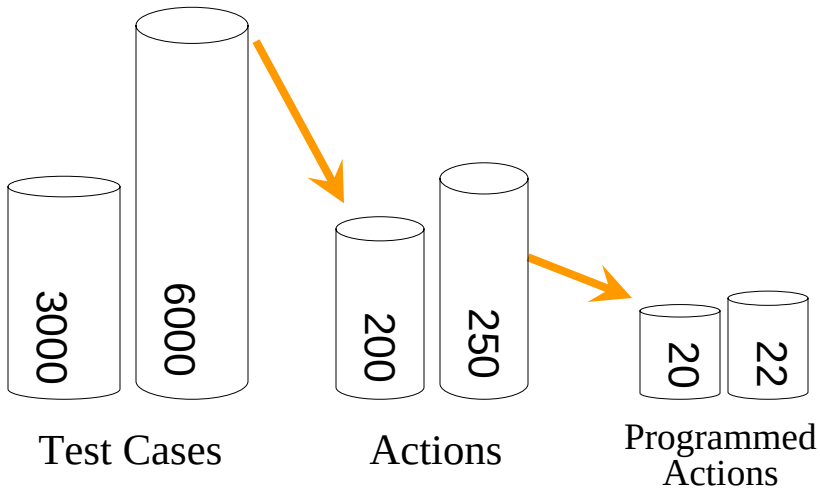
```
SelectLine(argument(1))
captured = CaptureField("Product Type")
Check( argument(4), captured)
```

- If "result" is equal to "AAX", the check has "passed", otherwise it has "failed"

Automation Overview



Doing the Count Down (example)



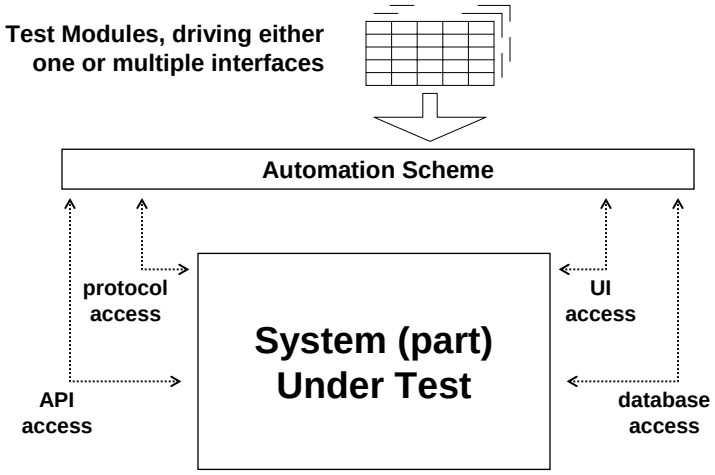
Some Tips to Get Stable Automation

- Make the system under test automation friendly
  - developers are not always motivated to do that, but it pays off
  - in particular ask to add specific property values to the GUI interface controls for automated identification
    - like “accessible name” in .Net and Java, or “id” in Web controls
- Pay attention to timing matters
  - in particular use “active timing”, based on the system under test, not fixed amounts of “sleep”
- Test your automation
  - develop a separate test set to verify that the actions work
  - make separate people responsible for the automation
- Use automation to identify differences between versions of the system under test

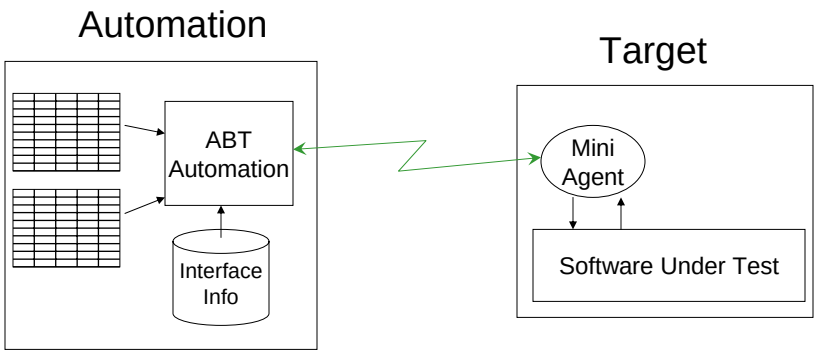
Non-UI Testing

- Examples
  - application programming interfaces (API's)
  - embedded software
  - protocols
  - files, batches
  - databases
  - command line interfaces (CLI's)
  - multi-media
- Impact is mainly on the automation
  - test design should in most cases be transparent towards the specific interfaces

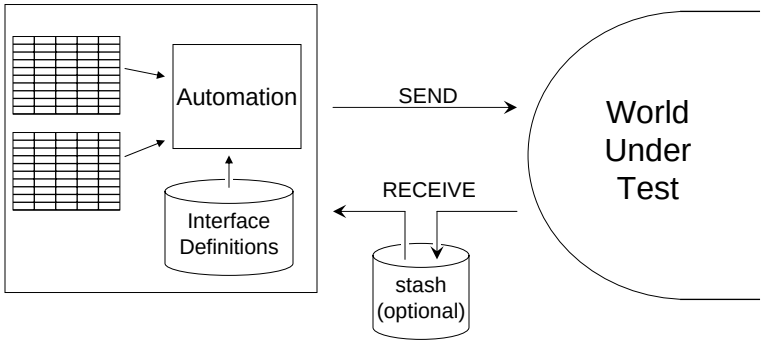
Multiple System Access



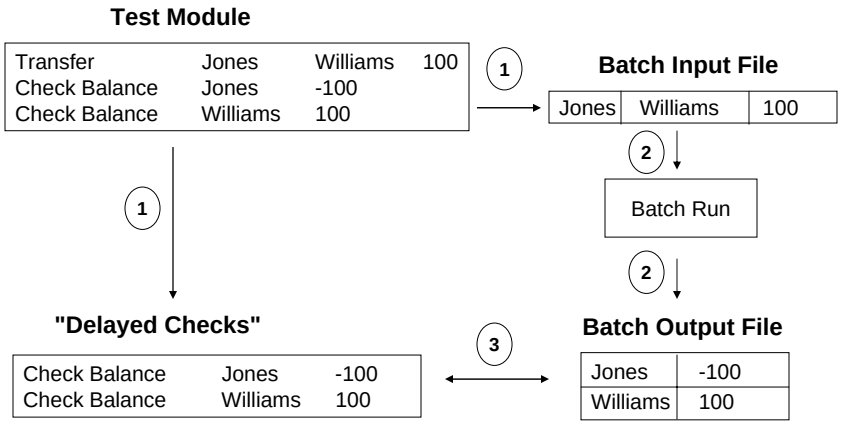
Testing Embedded Software



Testing a Protocol



Handling Batch Processes



Handling Multi-media

- Video, sound, pictures, in various formats
- Tests involving multi-media usually need to be treated as non-deterministic
  - even if behavior is not randomized: for example it might be theoretically possible to determine a bitmap or sound wave in advance, but it is usually not practical to do so
- Approach is highly dependent on the specific situation

Handling Multi-media, some strategies

- Consider manual testing
- Find criteria and objects that you CAN test
  - colors, volume levels, text recognition, etc
  - might be possible to convert formats to more accessable ones
- Investigate: Try to find handles that represent what is going on
  - file names, numeric handles, api calls
- Use a "play list" approach to reduce human testing

The "Play List" Approach

- The testers focus on a play list
  - items are played/shown while the machine tells what it is the tester should see/hear
  - the tester acknowledges/falsifies
  - the system stores checksums of those items that were ok
    - efficient and relatively easy to store and handle
  - Checksums can be kept for files, bitmaps etc
- Next time only the changed items go on the play list again
  - when the checksum is different from the previous run/version
  - in most systems this will only be a limited amount of items

Performance Testing

- The topic is complex, many components and phenomena
- To formulate tests in actions can be surprisingly straightforward
- Often performance testing isn't testing, but more close to research
- The three controls you can/should address:
  - hardware (equipment, infrastructure, etc)
  - software (programs, database models, settings, etc)
  - demands (1 second can be 10 times more than 2 seconds)

See also: "Web Load Test Planning", Alberto Savoia, StickyMinds.com

Some Tips to Get Stable Automation

- Make the system under test automation friendly
  - developers are not always motivated to do that, but it pays off
  - in particular ask to add specific property values to the GUI interface controls for automated identification
    - like “accessible name” in .Net and Java, or “id” in Web controls
- Pay attention to timing matters
  - in particular use “active timing”, based on the system under test, not fixed amounts of “sleep”
- Test your automation
  - develop a separate test set to verify that the actions work
  - make separate people responsible for the automation
- Use automation to identify differences between versions of the system under test

Test Execution

Execute Test | Repair Bugs | Execute Test | Repair Bugs | Execute Test | Repair Bugs | Execute Test | Repair Bugs | Execute Test | . . .

Communicating the Status

Is the system ready?

Test Environments

- Physical
  - hardware
  - infrastructure
  - location
  - . . .

  → costs money, can be scarce
- Software
  - programs
  - data models
  - protocols
  - . . .

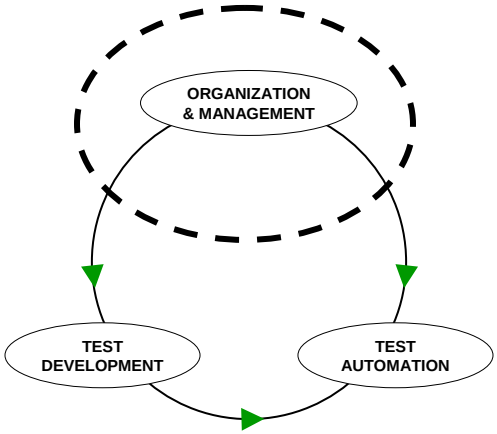
  → version management, time lines
- Data
  - initial data
  - parameters / tables
  - . . .

  → availability, manageability

"Test Execution Heaven"

- In control of the test environment
- Multiple separate test databases
- Possibility to reload a test database quickly to a standard initialization
- Being able to modify system dates
- In control of couplings
- Good response times
- Available timely
- Right versions of developed/changed products

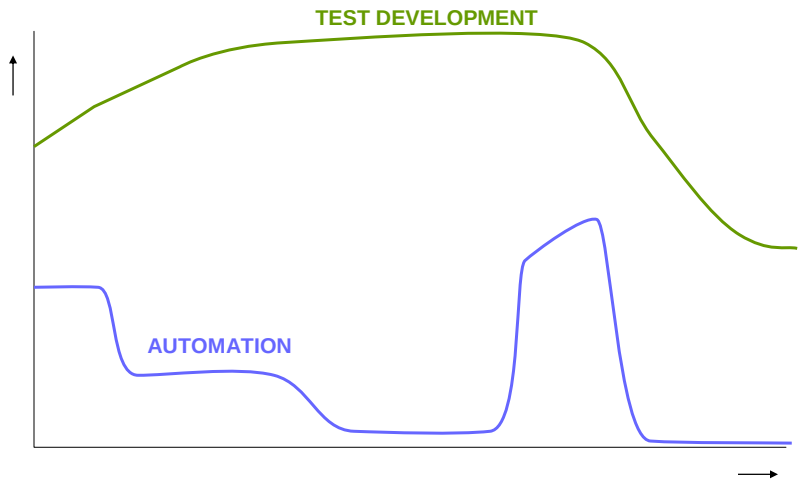
Topics Test Management



Plan of Approach (Example)

- Test assignment
- Approach
- Activities
- Products
- Test Environment
- Planning
- Risks
- Project organization
- External dependencies
- Costs and benefits

Typical Time Allocation



Assembling the Team

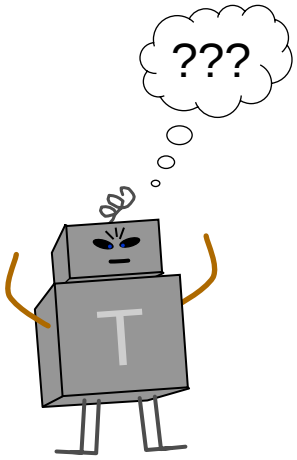
- Within a project or QA group
  - Test management: managing the test process
  - Test development: production of the test modules
    - Test Leads
    - Test Developers
    - End Users, Business Analysts
  - Automation Engineering: production of the automation scheme for automatic execution
    - Lead Engineer
    - one or more Automation Support Engineers
- General, across projects
  - Support Function, providing
    - methods, techniques
    - know how
    - training
    - tools
    - environments

Embedding QA

- Clear division of tasks and responsibilities:
  - testers (who will now become “test developers”)
  - users, matter specialists
  - test automation engineers (do they work for you?)
  - developers
  - other stakeholders
- Decision making and communication
  - who's call is it to pass a system for release
  - what happens with bugs
  - who does which testing
  - what whistles to blow/not blow, when
- Flexibility
  - testing is a "2nd order dependency" activity
  - firmness on the integrity of provided information (but be practical)

Question

- What are good skills for a tester?

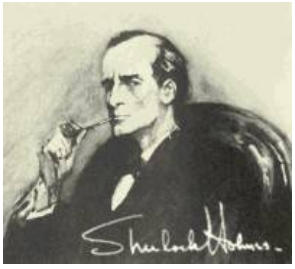


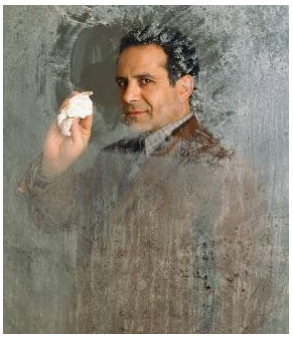
Skills for the Team

- Typical skills needed
  - testing (real testers)
  - subject matter
  - technology (often underestimated and then weak spot)
  - engineering practices (like configuration management)
  - soft skills (communication, managerial, diplomacy)
- Acquiring skills
  - training, reading, practicing, coaching
  - interest, curiosity
  - hiring, borrowing
  - overall picture counts, not everyone needs to know everything
- Standards and guidelines
  - helpful if practical, need constant attention

The Tester as a Detective

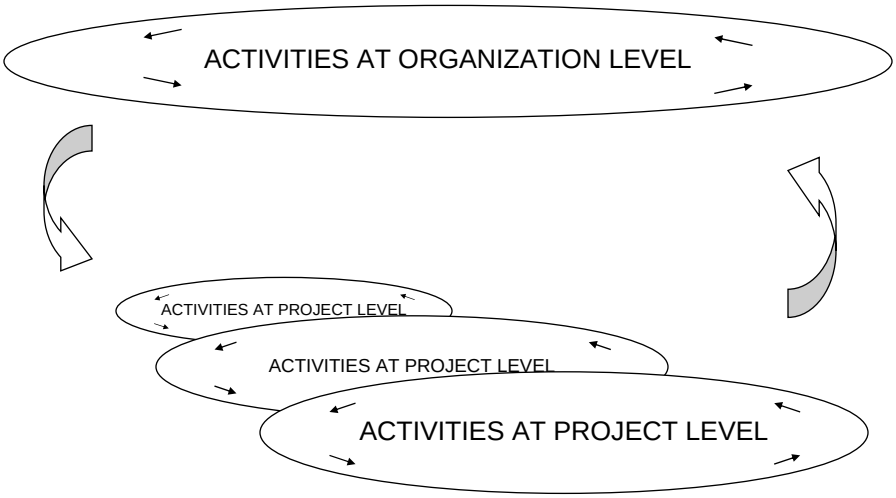
- Looking for the bad guys (bugs)
- Different personalities can be good testers



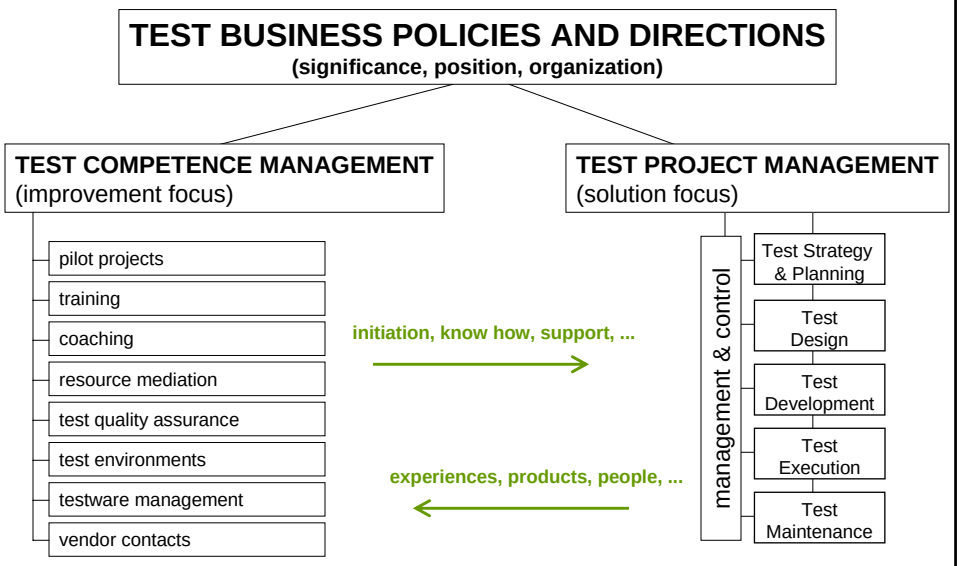


Tester Needs Hunting Instincts

Organization and Project Level



Beyond the projects: Test Governance



Test Strategy and Planning

- Typically related to project level
- Establish what to test, why, how, by who and when
- Discussion starts with stake holders
- Relate it to the test module breakdown
- Often underestimated, also by testers

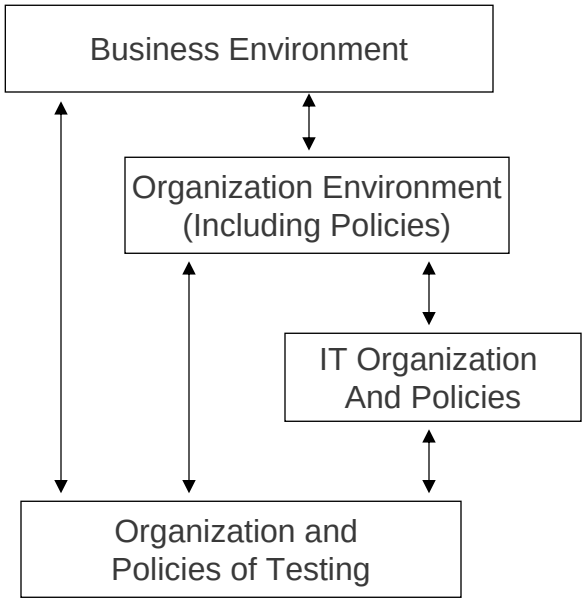
Test Policy

- Corporate point of view on testing (or software quality)
- Not a common topic in many organizations
- However can be useful to have

Examples of Policy Statements

- Testing and test automation are regarded as business critical
- Testware (developed tests and automation) is to be treated as a business asset
- User friendliness is a major priority in testing
- Test Automation is used as much as possible
- QA is responsible for timely and accurate information on system quality, independent of other stakeholders
- System Development is responsible for system quality
- Releasing a system is a business responsibility

Policy Model



How to Get It

- Think about it yourself
- Identify the stake holders
- Use existing policy documents (if any)
- Discuss it:
  - Within the testing department
  - Externally (IT, “business”, stakeholders)
- If possible start low profile and end high profile
- Policies should be recognized and understood
- Consider the use of an “incremental” document

Typical Issues (1)

- What is the significance of testing and what may it cost
- Connection to critical success factors of the company
- Do we have problems
- When should testing be done in the life cycle
- Who should be involved (test development, assessment, reporting)
- Who is responsible

Typical Issues (2)

- How to deal with testing expertise
- Centralized / decentralized
- Methods and tools
- Migration to a desired situation
- Automation or not

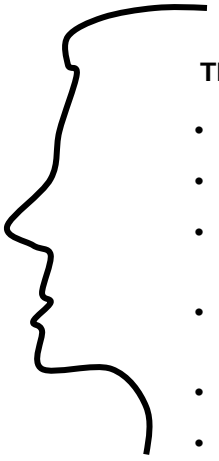
Project Management: "Testing in the Cold"

- Resistance
- Commitment
- Politics
- Dependence
- Managing expectations
- Difficulty
- Motivation
- Practical issues and problems

Resistance is Normal

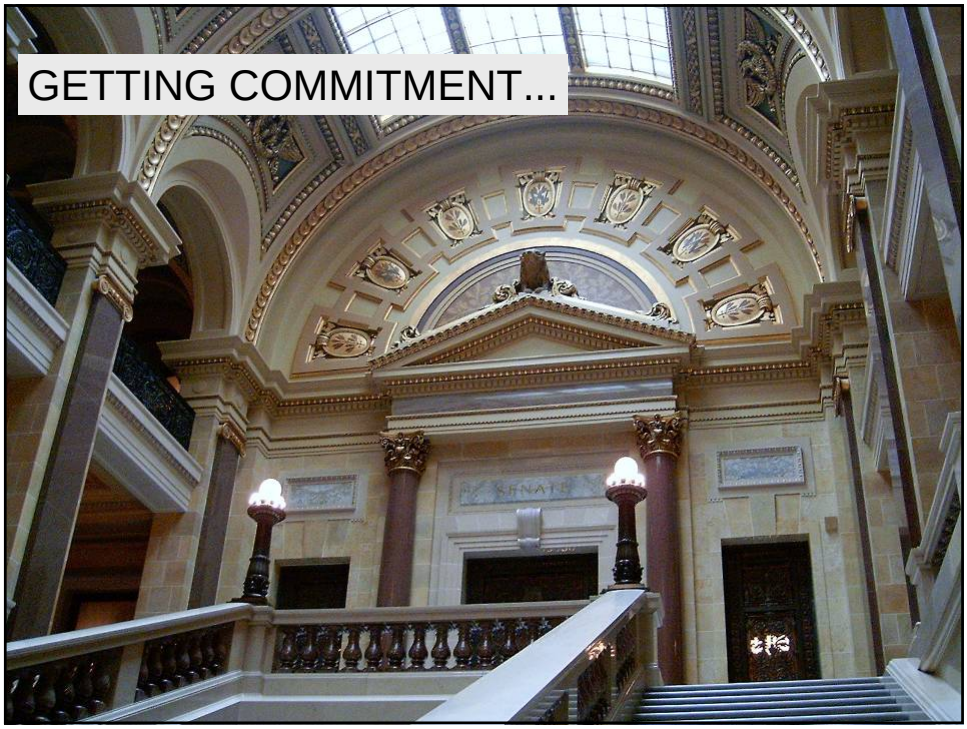
WHEN THEY <u>SAY</u> :

- *"Let's reconsider"*
- *Now is not the time*
- *All the time new objections*
- *"Fine, but are we ready for this?"*
- *Saying nothing*
- *Saying yes, acting no*
- *The method is good, but in this specific case ….*



THEY COULD <u>THINK</u> :

- *"I don't understand this"*
- *"I didn't expect all this"*
- *"This is going to cost me my job"*
- *I'm the star here, I don't need the competition*
- *"We can't achieve this"*
- *We will become too dependent on those guys*
- *Now they will find out how bad we are testing*



Keeping Management Commitment

Testing is often not popular . . .
Nobody wants an extra task (= problem)

- Offer solutions, not additional problems
- Tell managers that a good tested system creates a positive image (not only negative reasons for testing)
- Present/show what you're doing (glass box)
- Try to get clarity about policies and directions for testing
  - see the slides earlier in this presentation
- Keep in mind: managers want things to be under control
  - give clear and timely information about (1) progress and (2) results
- Use outsiders and/or books/articles to make your case
- Try to find some bugs . . .

Commitment, Specifics

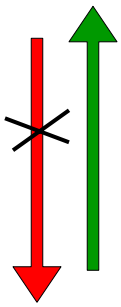
- "No time, no money, ..."
  - Back to the problem
  - You should not become the problem owner!
- "It is so expensive/it is so difficult"
  - Testing is expensive and difficult
  - Test automation is difficult
- "The others should do the testing"
  - Figure this out
  - You can't deal with this yourself
- General vagueness
  - Hidden problems and conflicts

Politics . . .

Who is to blame when things go wrong
Systems under test are sometimes presented as working, while they're not yet finished

- Notify the responsible managers early about what is going to happen
- Make clear written down procedures, especially for the test execution phase
- Make the test process transparent
  - example: use a "health check module" to visualize that the underlying system isn't working
- Maintain an atmosphere of cooperation and communication with all
- Ask for help, sound the alarm,
  - don't underestimate the "politics of failure", it can hurt you

An (Old) Joke, Still Valid



The Life Cycle of an IT Project

- Enthusiasm
- Disappointment
- PANIC ! ! !
- Looking for the guilty ones ("blame-storming")
- Punishing the innocent
- Rewarding those who had nothing to do with it

Dependencies . . .

Testing and automation is dependent on many factors:
- Working systems
- Test environment(s)
- People
- Specifications
- ...

- Make clear arrangements with everybody about everything, as early as possible
- Keep in touch with the rest of the project
- Make the high level test products as early as possible

Dependencies, specifics

- "The system under test isn't available"
  - The automation will be the first in trouble
  - Discuss this (early)
  - Allocate resources when really needed
- "The system under test doesn't work"
  - The automation will be the first in trouble
  - Create a "health check" (smoke test to see if all functionalities that are relevant to the automation work)
- "There is no test environment"
  - Start planning this as early as you can
  - Make clear how important this is
  - "No test environment, no test"

Expectations Around Automation

The impression can arise that testing is just a push of a button (especially when test automation is involved)

. . . even if nobody said so!

- Message: testing is difficult
- Make clear what is happening
- Communicate, manage expectations
- Put high figures in your test plan (especially for the test execution),
  - at least don't put too low figures, let somebody else lower them
- Ask managers and other people involved what they expect

Difficult to Keep People Motivated

Motivation of team members can erode when time progresses
Happens to both test developers and automation engineers

- Watch the signals
- Make sure that the right people are assigned
  - Navigators should have a programming background
  - Analysts should have a business background
- Avoid “monks work” (Dutch expression for repetitive work)
- Differentiate the work
- Avoid isolation of the test group
- Be prepared to switch people
- Worry about this, and arrange solutions, in advance
- Create operational and professional communication structures
  - for example special interest groups to regularly discuss professional topics

Complexity: The Work is Difficult

Making tests should be difficult (if not, you do it wrong...)
- Finding bugs of others
- Making concrete examples

Automating them is difficult as well

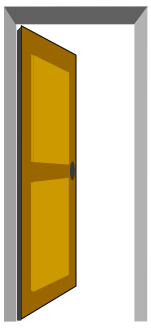
- Use coaching from experienced peers or outsiders
- Use reference visits to other projects or sites
- Create special interest groups
- Keep in touch with others doing the same work
- Organize interactions (meetings!), for example with:
  - users
  - developers
  - auditors
- Delegate tasks (better lazy then crazy...)

Practical Issues and Problems

Does the test tool work here ? ?
Do we have authorizations on the test environment ? ?
Where do we keep our test products ? ?
Questions, questions, . . .

- Be prepared for this, automated testing is a “cloud of details”
- Put suitable team members in leading roles, and delegate responsibilities to them
- For larger test projects test management is a full time job, make sure you get this time
- Try to find easier ways, organize things better, or automate more
  - “If you have a difficult task, ask a lazy man. He will find an easier way”

Getting Started . . .



- Treat introduction of automation or automation improvements as an organizational change
  - not necessarily a re-organization, but a different way of working
  - no longer just “testing”, but also “test development”
- No standard recipe, depends on:
  - skills available
  - experience with testing and test automation
  - time (and budget) constraints
- Use pilots, training and coaching
  - internal or external
  - some form of ongoing coaching is essential, there are many lessons to learn
- If possible try to make a start, don’t wait
  - the “right moment” for process improvements might never come, things are always busy

Homework . . .

- Brian Marick, New Models for Test Development, www.testing.com/writings/new-models.pdf, 1999
- Cem Kaner, Hung Nguyen and Jack Falk, Testing Computer Software 2nd Edition, Wiley, 1999
- Cem Kaner, James Bach, and Bret Pettichord, Lessons Learned in Software Testing, 2001 (expected)
- Dorothy Graham and Mark Fewster, Automating Software Testing, Addison Wesley, 1999
- Hans Buwalda, Action Figures (on model-based testing), STQE Magazine, March 2003
- Hans Buwalda, and Maartje Kasdorp, Getting Automated Testing Under Control, STQE Magazine, November Issue 1999
- Hans Buwalda, Dennis Janssen and Iris Pinkster, Integrated Test Design & Automation, Addison Wesley, 2001
- Hans Buwalda, Soap Opera Testing (article), STQE Magazine, February 2005
- Hans Buwalda, Soap Opera Testing (keynote), STAR East 2000
- Hans Buwalda, Testing with Action Words, Abandoning Record and Playback, Eurostar 1996
- Hans Buwalda, The Three Holy Grails of Test Development, Quality Week 2001
- Hung Nguyen, Michael Hackett, Brent K. Whitlock, "Happy About Global Software Test Automation", Happy About, 2006
- Hung Nguyen, Robert Johnson and Michael Hackett, Testing Applications on the Web 2nd Edition, Wiley, 2000