Acceptance Test Driven Development

Applicable in both V-model and Agile

Chris Schotanus, Meile Posthuma & Maurice Koster

Test First, Build Next

V 2014.1.0.8

Experience the commitment®

Acceptance Driven Development - Agenda

- Introduction of (Acceptance) Test Driven Development
- Workshop
- Demo of Tooling used with ATDD
- Evaluation

CGI

Introducing Test Driven Development

CGI

Experience the commitment®

eXtreme Programming's "Test First" principle

- Create test cases before writing code
  - Business representatives write acceptance tests to demonstrate that user stories are correctly implemented
  - Programmers continually write component tests which must run flawlessly for development to continue
- "We only write new code when we have a test that doesn't work"

Quotes from Important Testers

"More than the act of testing, the act of designing tests is one of the best defect preventers known … The thought process that must take place to create useful tests can discover and eliminate problems at every stage of development" *Boris Beizer*

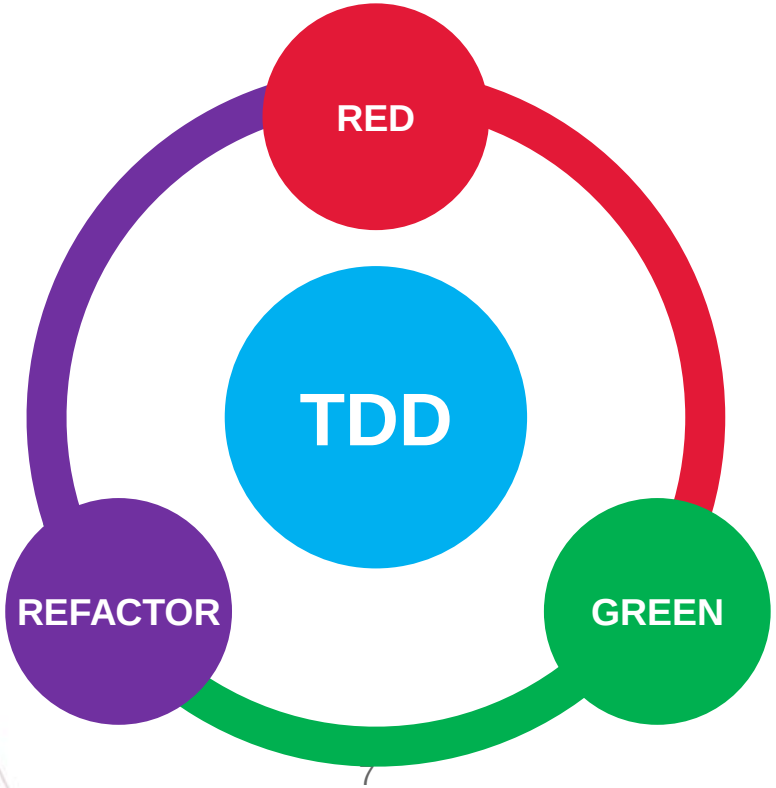
"One of the most effective ways of specifying something is to describe (in detail) how you would accept (test) it if someone gave it to you." *Bill Hetzel*

CGI

Introducing Test Driven Development

Test Driven Development (TDD) is a software development technique!

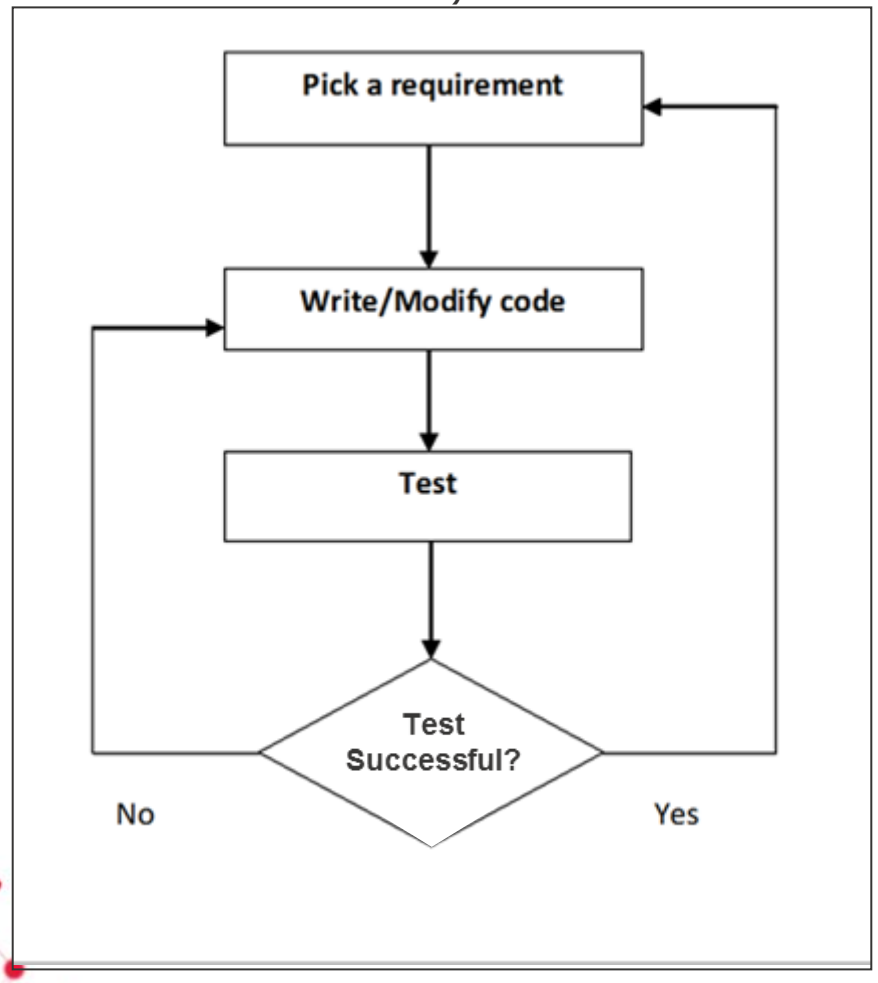
Although the technique is quite old, Kent Beck introduced the name in 2003. The principle is that the developer starts writing tests which are used to test the code that the developer will write.



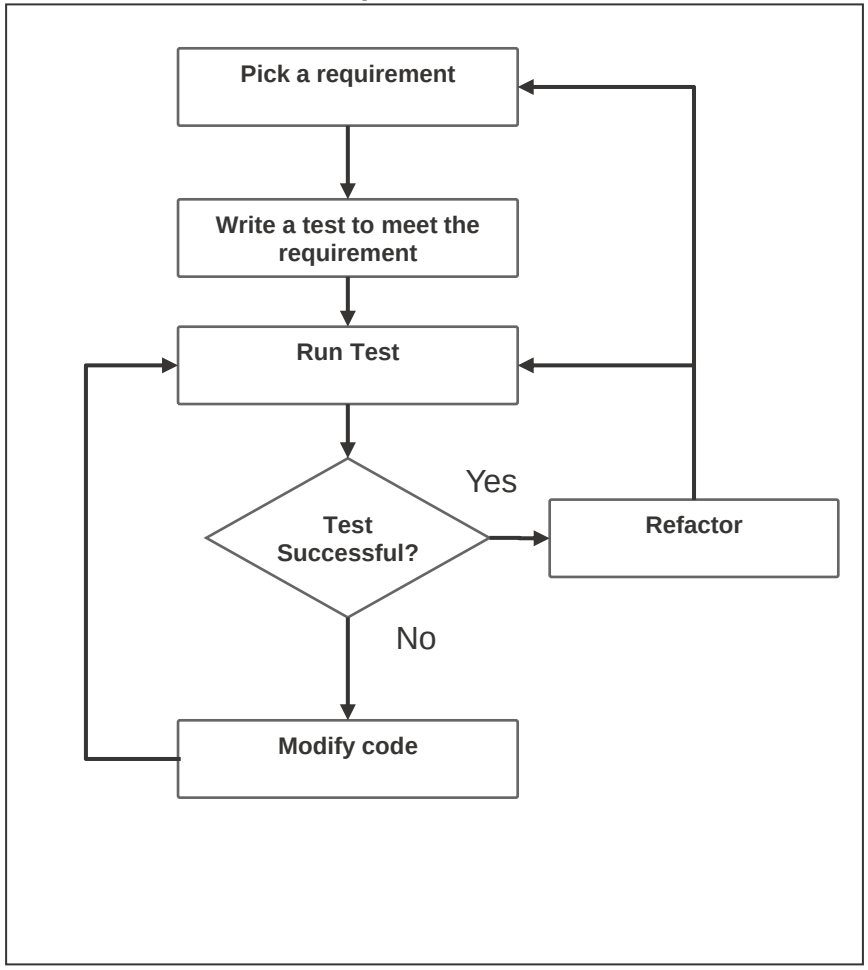
CGI

TDD compared with “Traditional Testing”

Build First, Test Next



Test First, Build Next



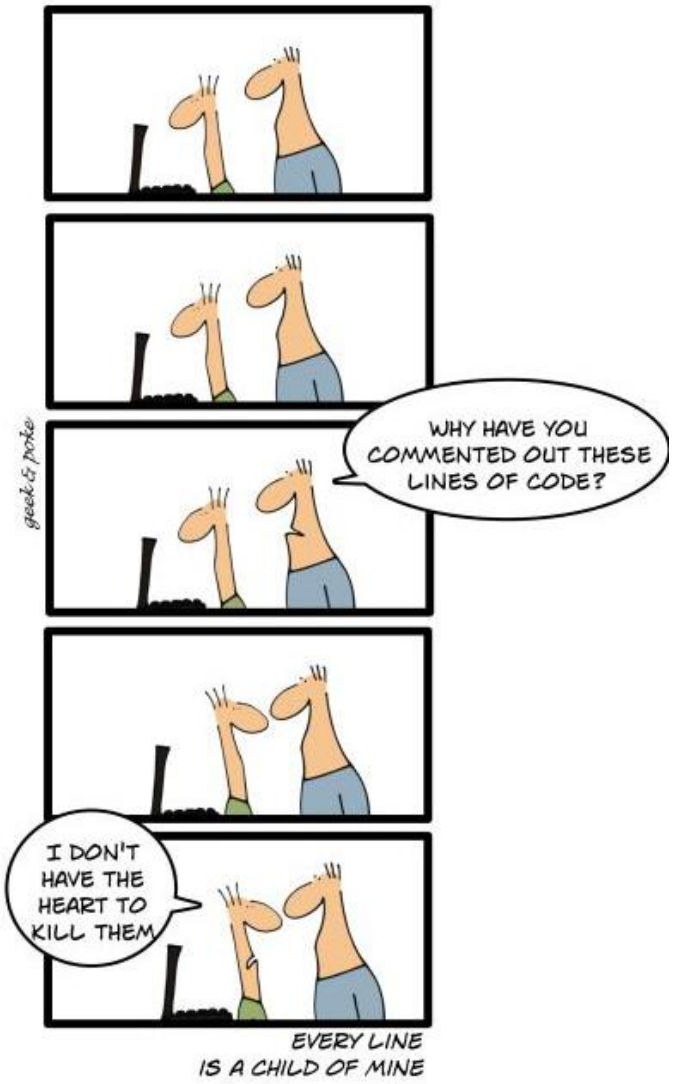
The goal of Test Driven Development

To write clean code that works, and for a whole bunch of reasons:

Clean code that works:

- is a predictable way to develop.
- gives you a chance to learn all the lessons that the code has to teach you.
- improves the lives of users of our software.
- lets your teammates count on you, and you on them.

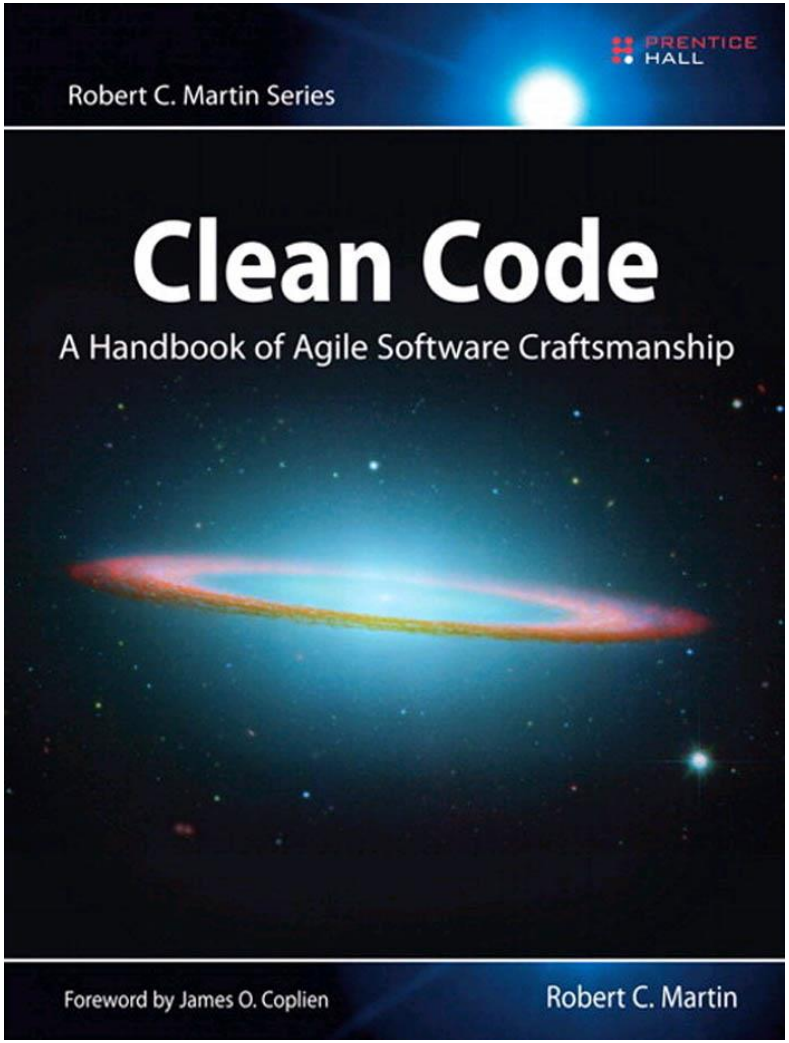
Writing clean code that works feels good.



From Kent Beck, Test Driven Development by example

Test Driven Development: “The Three Laws”

1. You may not write production code until you have written a failing component test.
2. You may not write more of a component test than is sufficient to fail, and not compiling is failing.
3. You may not write more production code than is sufficient to pass the currently failing test.



CGI

Advantages of TDD

- **Test Coverage**. 100% Statement coverage is implicitly reached.
- **Test Repeatability**. The tests can be run any time you like.
- **Documentation**. The tests describe your understanding of how the code should behave. They also describe the API. Therefore, the tests are a form of documentation.
- **API Design**. When you write tests first, you put yourself in the position of a user of your program's API. This can only help you design that API better.
- **System Design**. A module that is independently testable is a module that is decoupled from the rest of the system.
- **Reduced Debugging**. Debugging time is reduced enormously.
- **Your code worked a minute ago**! If you observe a team of developers who are practicing TDD, you will notice that every pair of developers had their code working a minute ago.

CGI

Some questions to answer

Which of the following is not one of the “thee laws” of TDD?

A – You may not write production code until you have written a failing component test.

B – You may not use tools to debug your production code in case of a failing test

C – You may not add comment lines in order to keep your code clean

D – You may not write more production code than is sufficient to pass the currently failing test

What is the correct sequence?

A – Write a test case, Write the code, Compile the code, refactor the code

B – Write the code, Compile the code, Write a test case, Run the code, Run the test

C – Write a test case, Run the test, Write the code, Compile the code, Run the test, Refactor the code

D – Compile the code, Write a test case, Run the test, Refactor the code

The correct answers

Which of the following is not one of the “thee laws” of TDD?

B – You may not use tools to debug your production code in case of a failing test

C – You may not add comment lines in order to keep your code clean

What is the correct sequence?

C – Write a test case, Run the test, Write the code, Compile the code, Run the test, Refactor the code

The correct answers

Which of the following is not one of the “thee laws” of TDD?

B – You may not use tools to debug your production code in case of a failing test

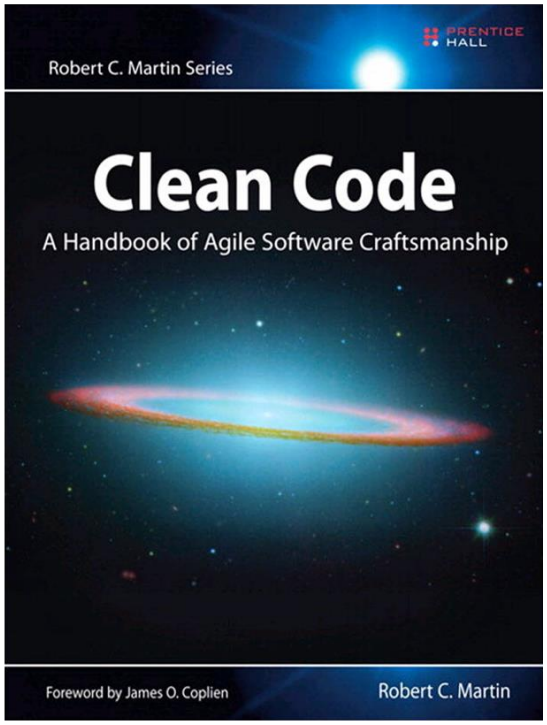
C – You may not add comment lines in order to keep your code clean

What is

C – Writ … the code, Run

Test Driven Development: “The Three Laws”

1. You may not write production code until you have written a failing component test.
2. You may not write more of a component test than is sufficient to fail, and not compiling is failing.
3. You may not write more production code than is sufficient to pass the currently failing test.



Acceptance TDD

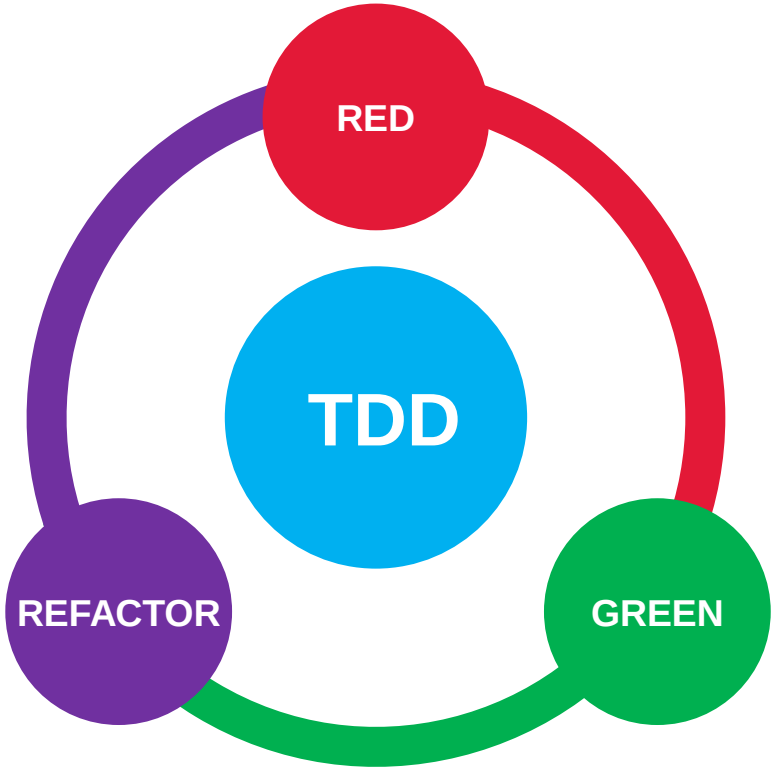
Aka Behaviour Driven Development

Aka Specification By Example

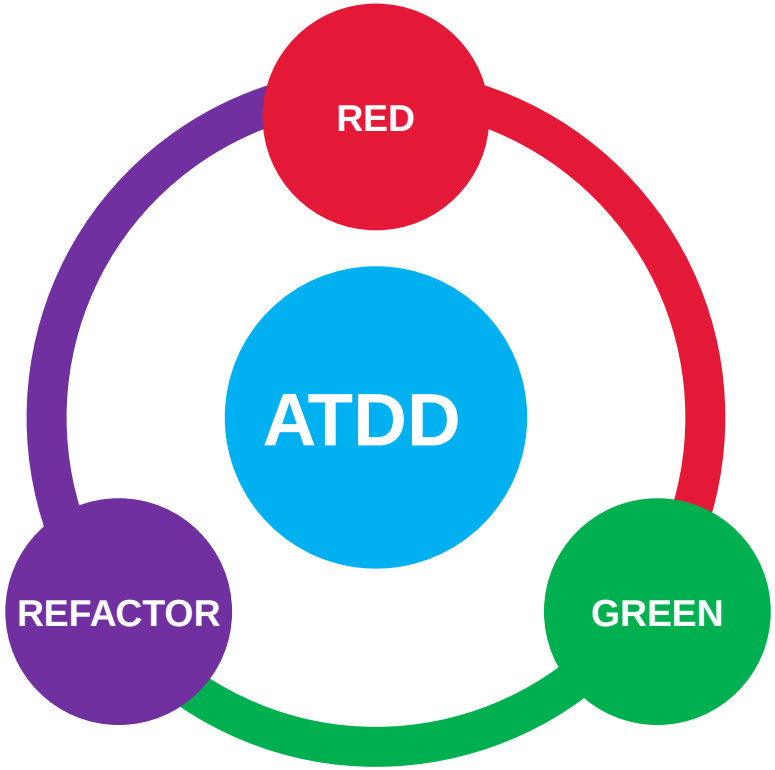
CGI

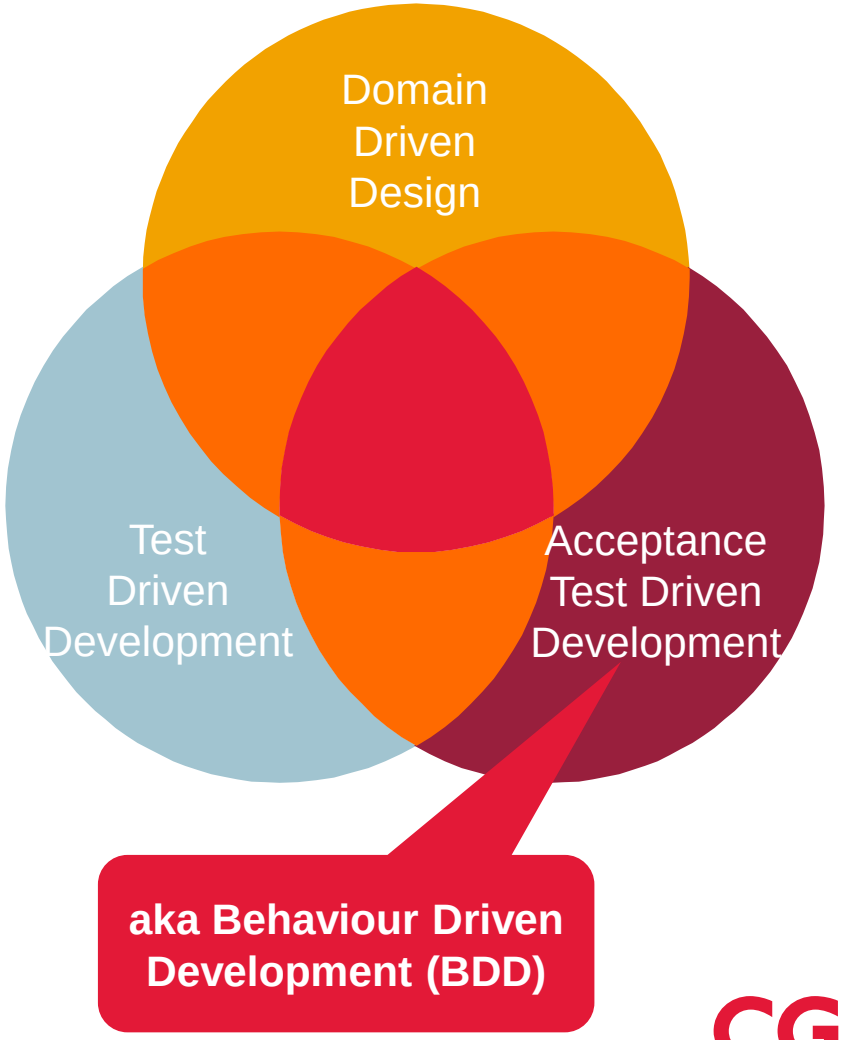
Experience the commitment®

Test first, build next!



Test first, build next!





Limitation of TDD

TDD leads to high quality software.

It's not guaranteed that it lead to the correct software

Test-driven development has proven that writing tests before coding can produce higher quality code. Still often customer requirements can be misunderstood.

CGI

Limitation of TDD

TDD leads to high quality software.

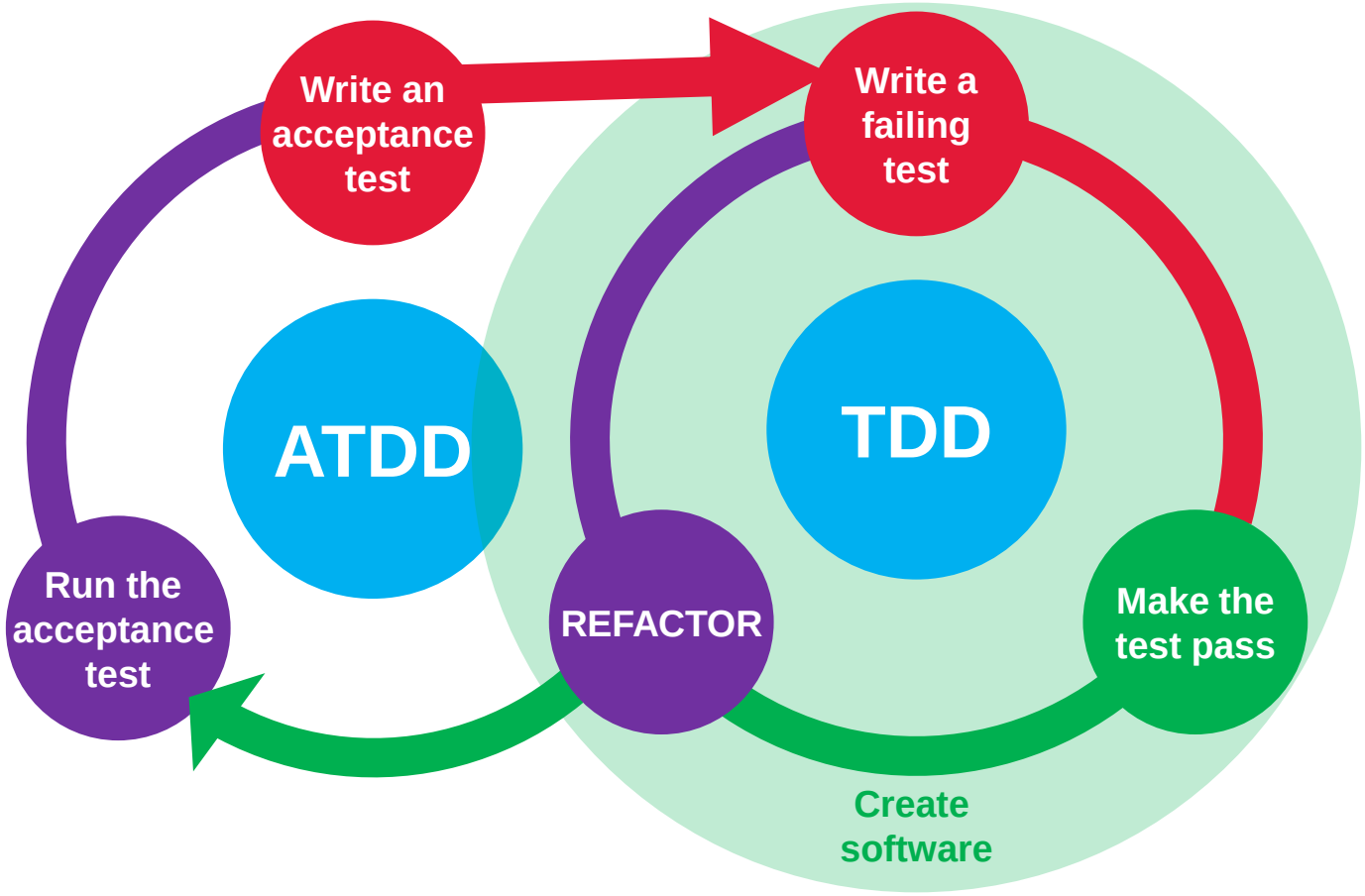
It's not guaranteed that it lead to the correct software

Test-driven development has proven that writing tests before coding can produce higher quality code. Still often customer requirements can be misunderstood.

A technique called acceptance test-driven development (ATDD) can reduce this risk

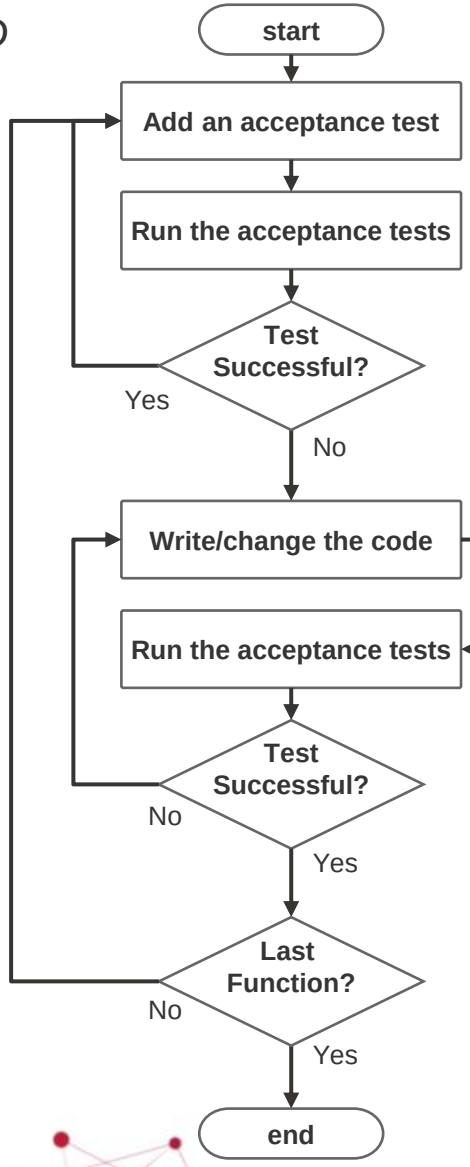
CGI

TDD combined with Acceptance testing

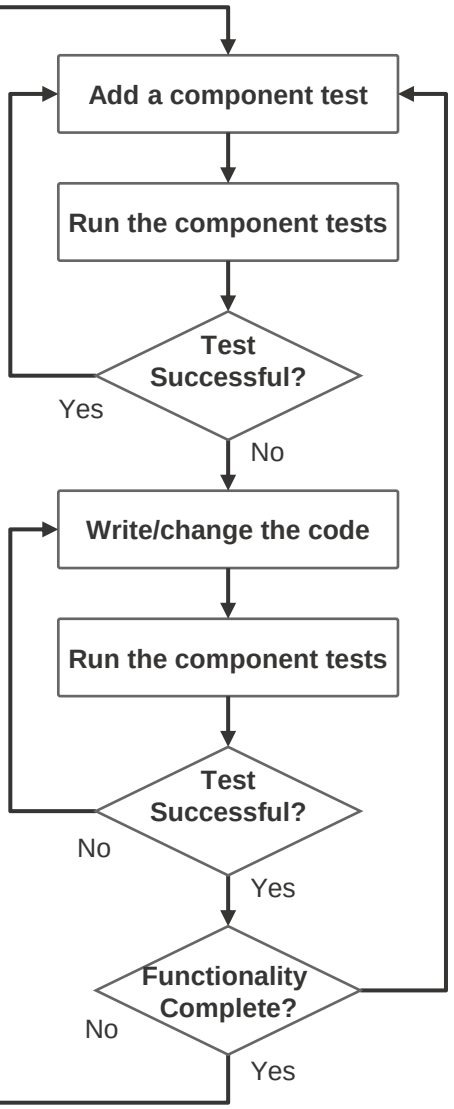


TDD combined with ATDD

Acceptance TDD

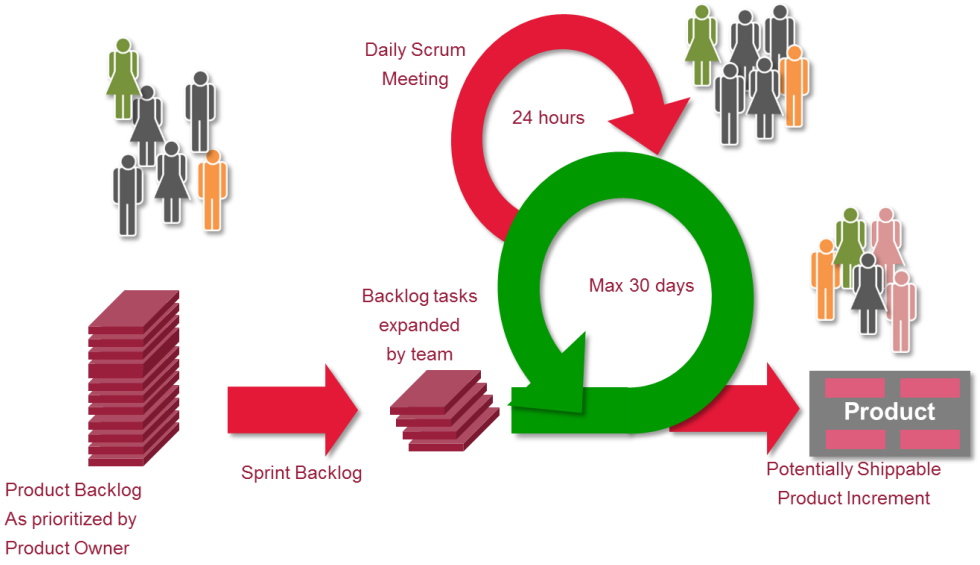


Developer TDD



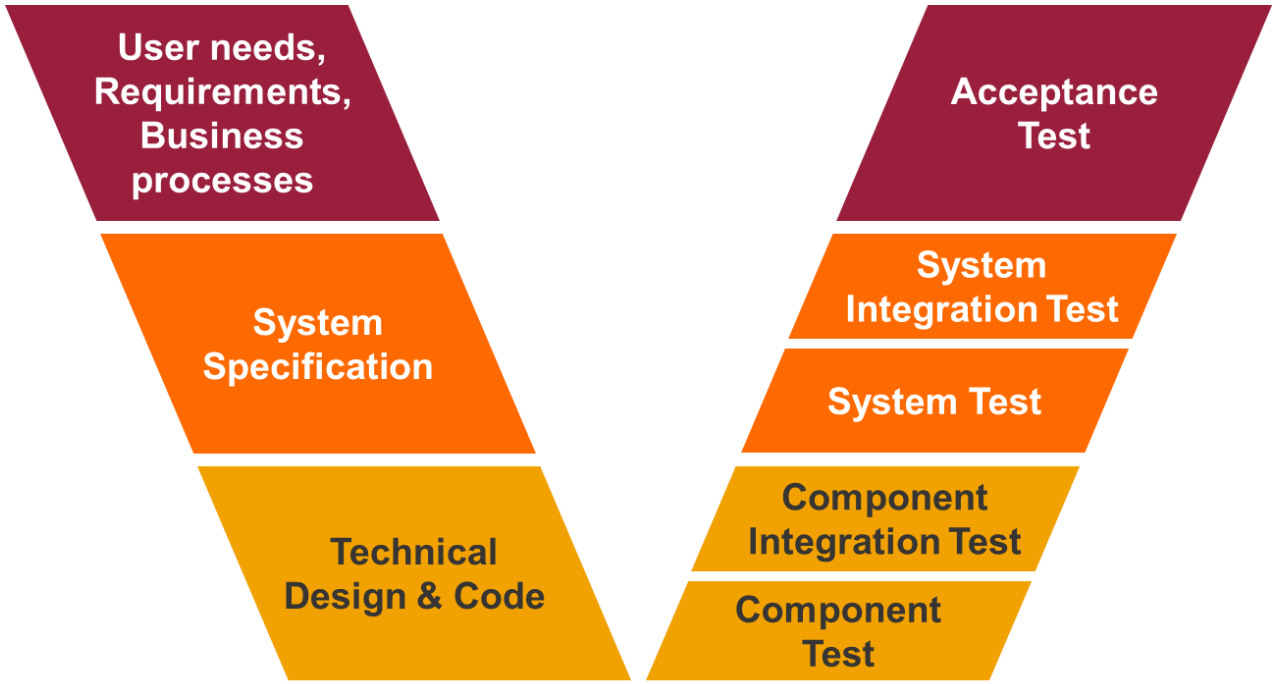
So, Why (Acceptance) Test Driven Development?

- One of the biggest problems in software is requirements ambiguity
  - A direct result of using natural language specifications (e.g., “The system shall be fast”)
- A test case is inherently unambiguous
  - Test cases are unambiguous “proxies” for requirements



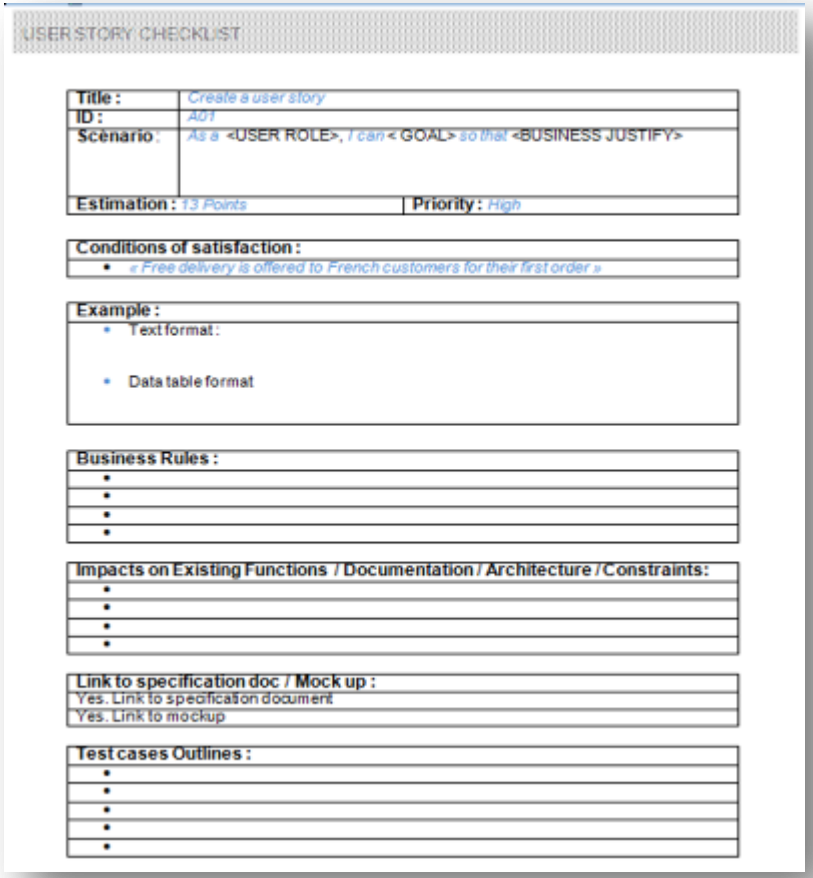
But in Linear development too

Adding (acceptance) test cases to the Request for Change (RfC) or UC will dramatically improve the quality of these documents and will reduce failures due to misunderstanding



Acceptance Test Driven Development

Using test conditions and test cases as design documentation



USER STORY CHECKLIST

Title :	*Create a user story*
ID :	*A01*
Scenario :	*As a* <USER ROLE>, *I can* < GOAL> *so that* <BUSINESS JUSTIFY>
Estimation : *13 Points*	**Priority :** *High*

Conditions of satisfaction :

- *« Free delivery is offered to French customers for their first order »*

Example :

- Text format :
- Data table format

Business Rules :

-
-
-
-

Impacts on Existing Functions / Documentation / Architecture / Constraints:

-
-
-
-

Link to specification doc / Mock up :

Yes. Link to specification document

Yes. Link to mockup

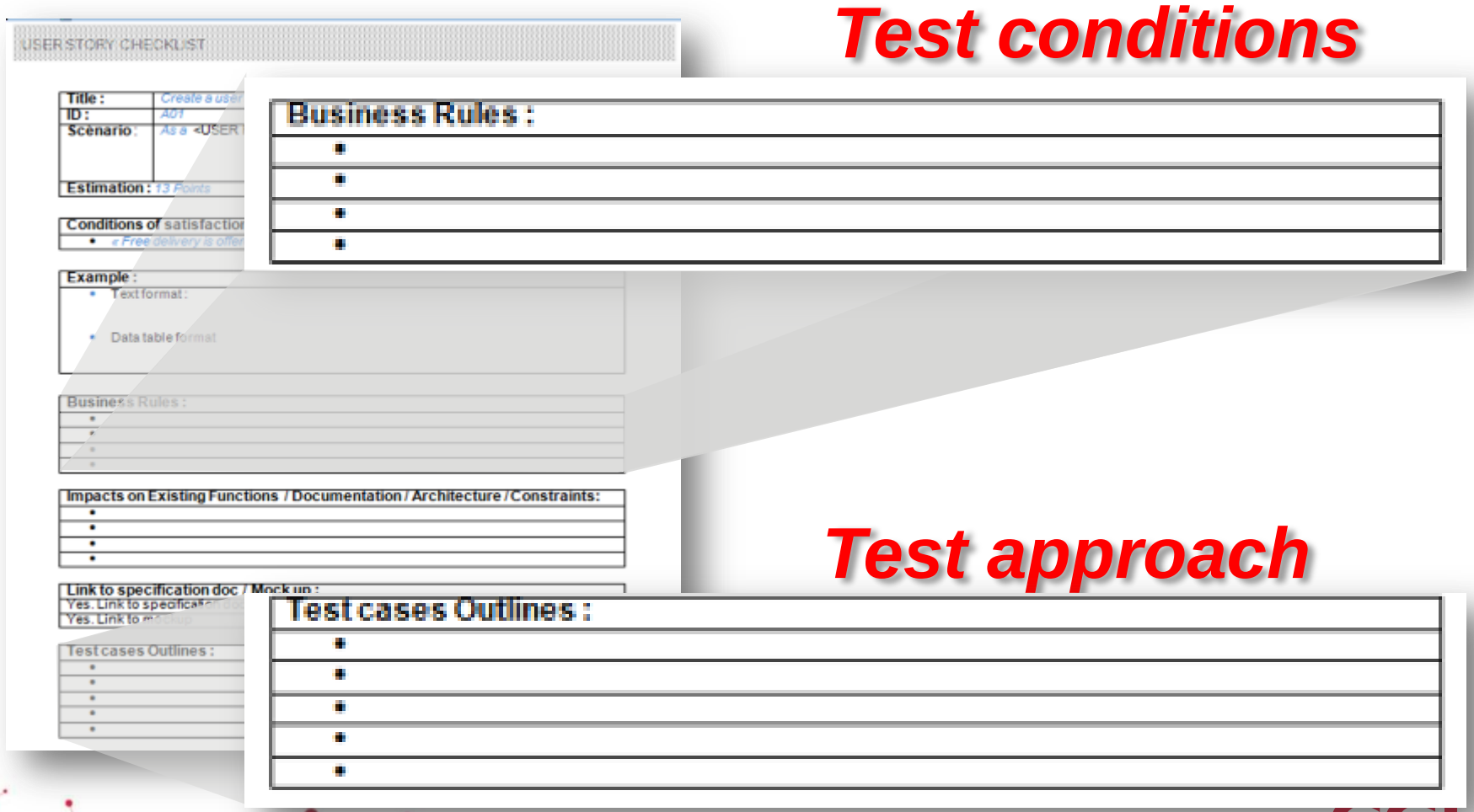
Test cases Outlines :

-
-
-
-
-

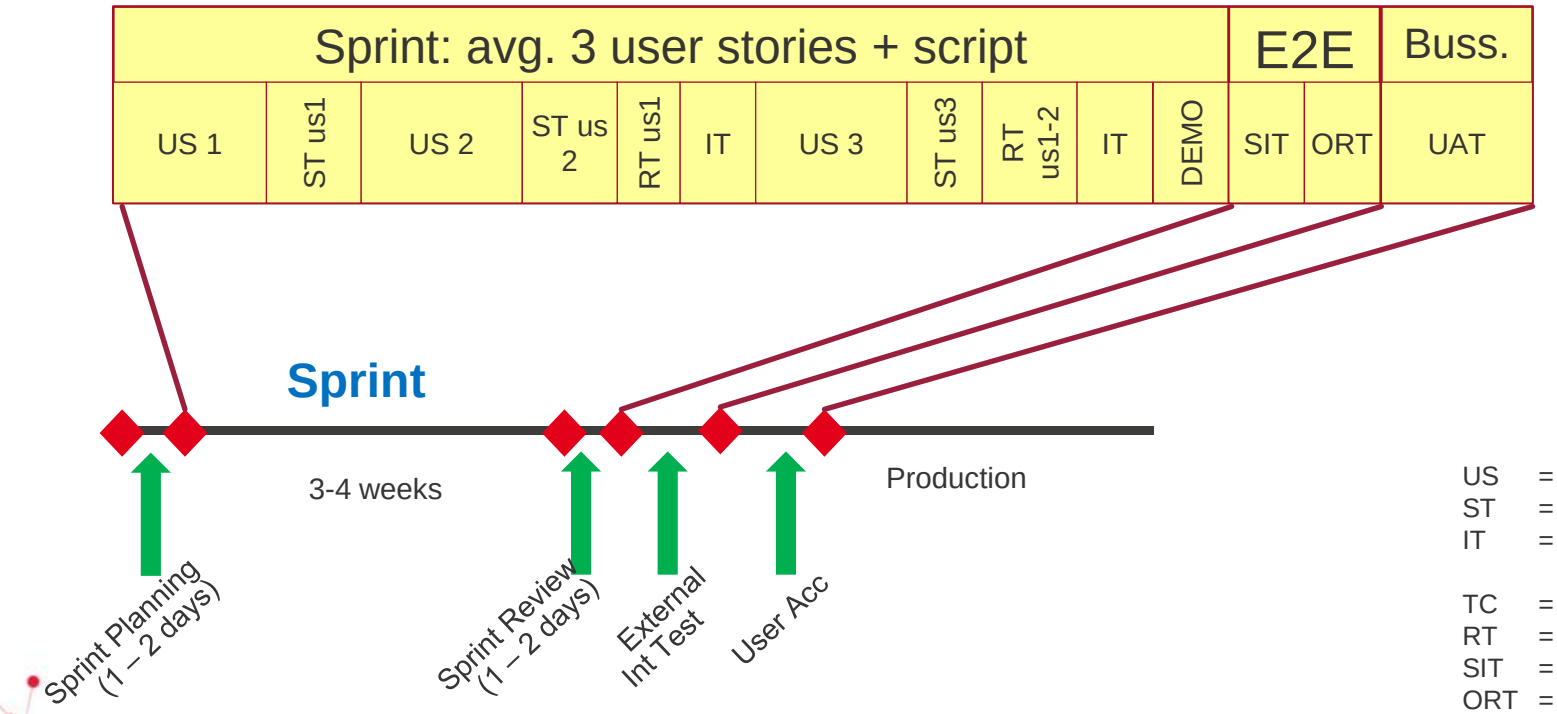
CGI

Acceptance Test Driven Development

Using test conditions and test cases as design documentation

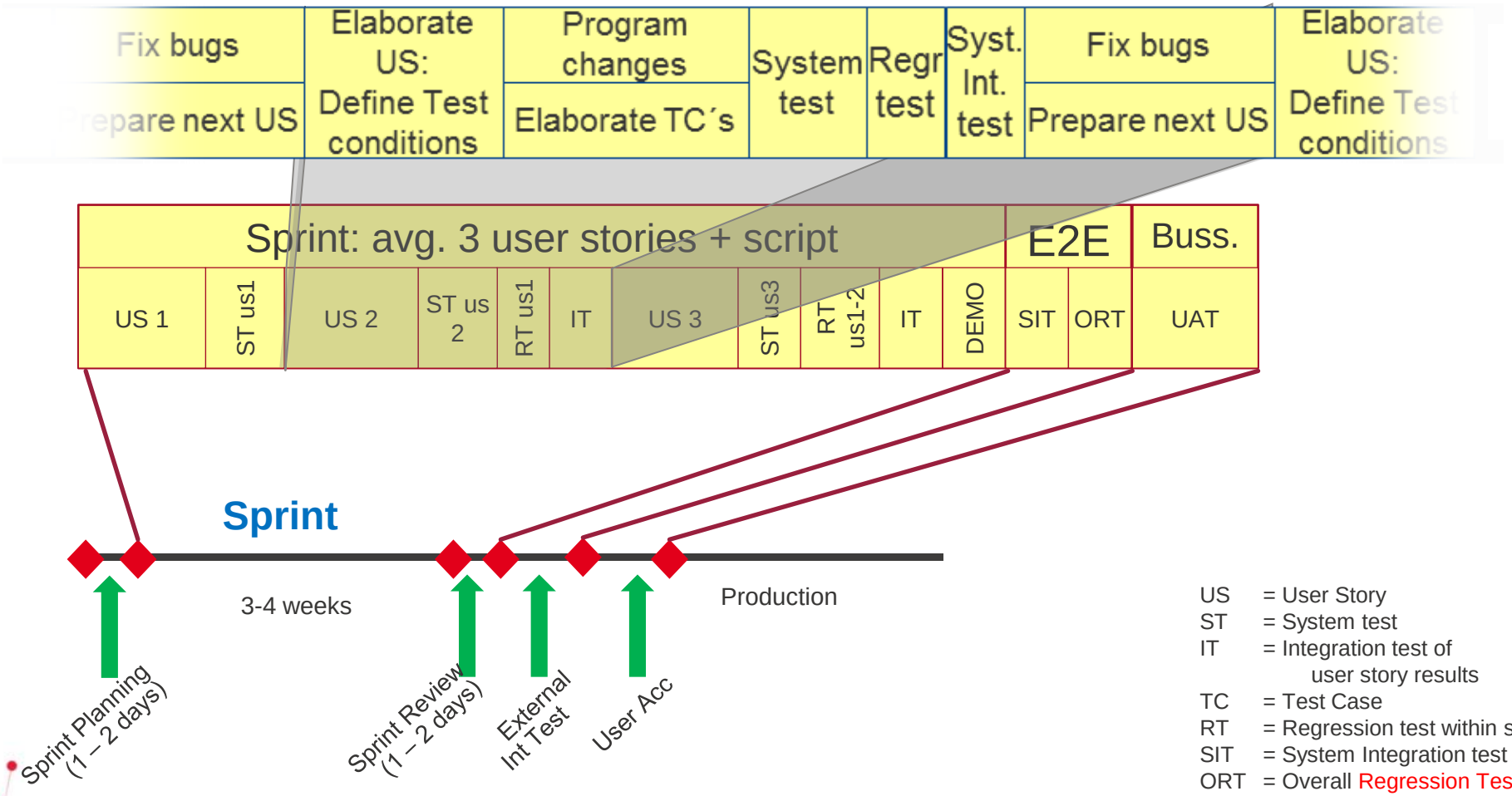


Testing in a sprint



US = User Story
ST = System test
IT = Integration test of user story results
TC = Test Case
RT = Regression test within sprint
SIT = System Integration test
ORT = Overall Regression Test
UAT = User Acceptance Test

Testing in a sprint



US = User Story
ST = System test
IT = Integration test of user story results
TC = Test Case
RT = Regression test within sprint
SIT = System Integration test
ORT = Overall Regression Test
UAT = User Acceptance Test

Some questions to answer

TDD does not guarantee correct software. That is because:

A – Programmers are not subject matter experts

B – Programmers just can’t produce correct software

C – At component level requirements may well be misunderstood

D – TDD Frameworks may contain bugs too.

Which of the following statements is correct

I – ATDD must be automated

II – TDD can be automated

A – I and II are correct

B – I and II are incorrect

C – I is correct, II is incorrect

D – I is incorrect, II is correct

The correct answers

TDD does not guarantee correct software. That is because:

C – At component level requirements may well be misunderstood

Which of the following statements is correct

I - ATDD must be automated

II – TDD can be automated

B – I and II are incorrect:

ATDD is an approach in which test cases are defined in advance. It is preferable but not necessary to automate the execution.

TDD is done with tools in the development environment and will always b eautomated.

The workshop

Part one, define acceptance criteria

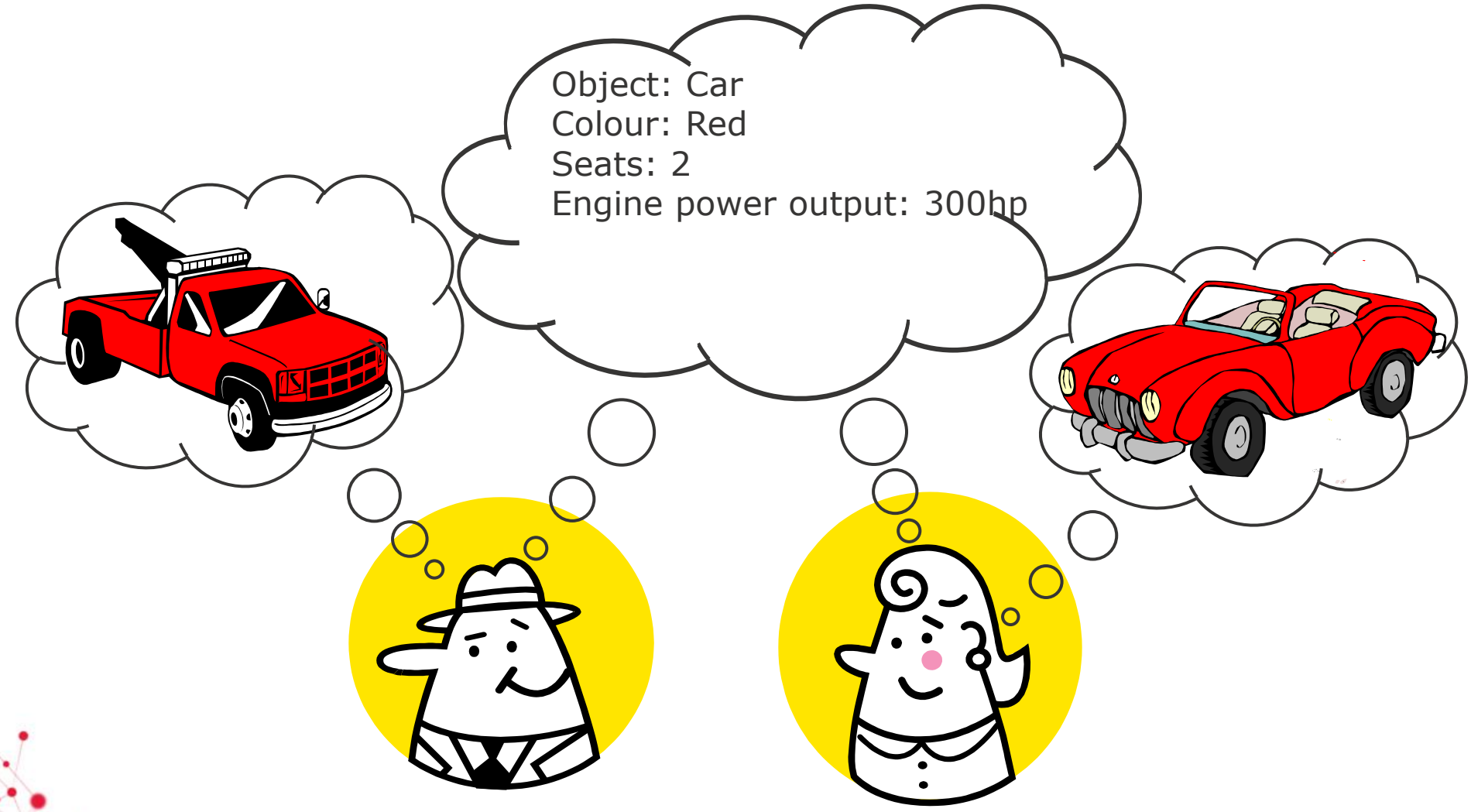
CGI

Experience the commitment®

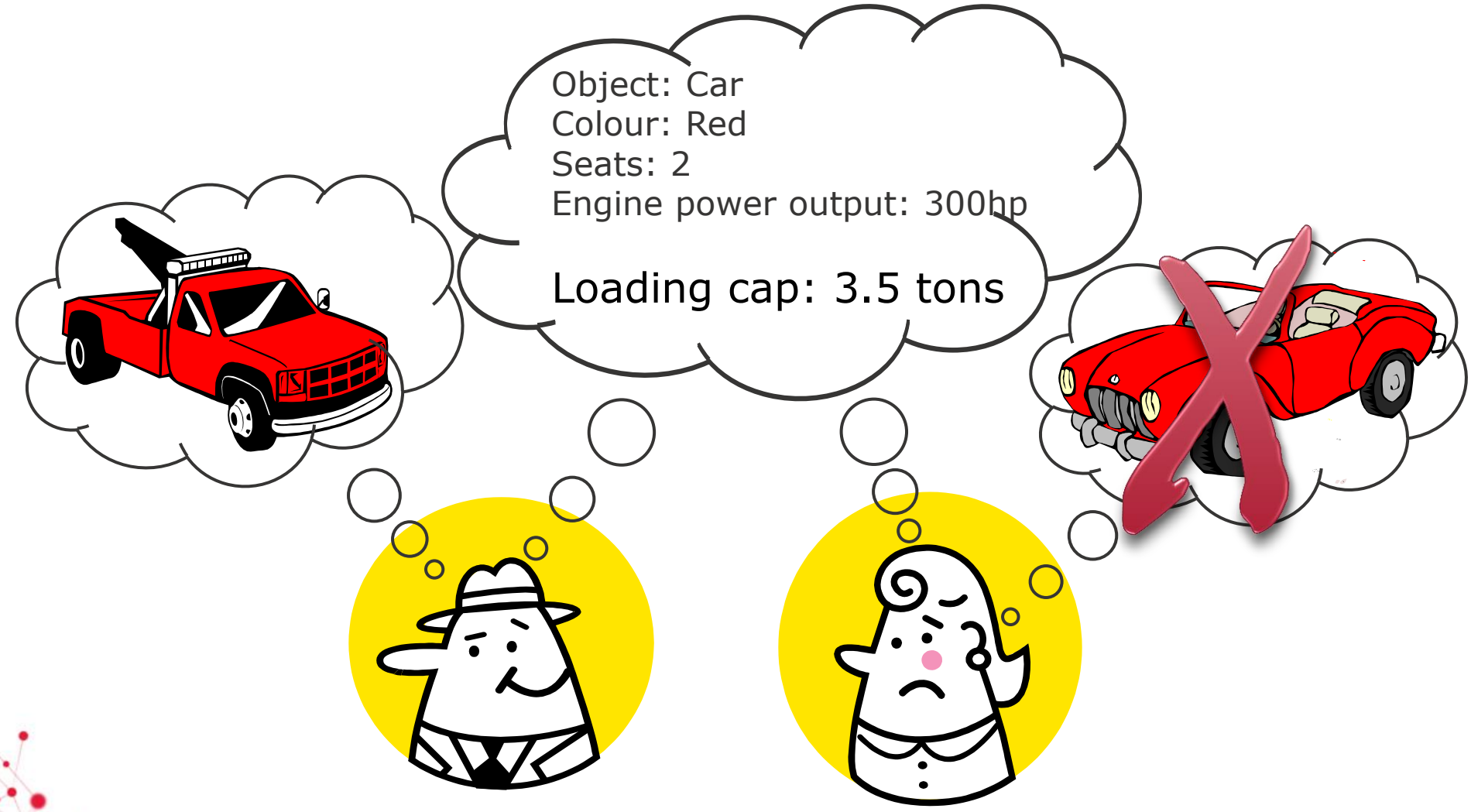
Defining ATDD test cases
The theory

Experience the commitment®

An example or requirement specification

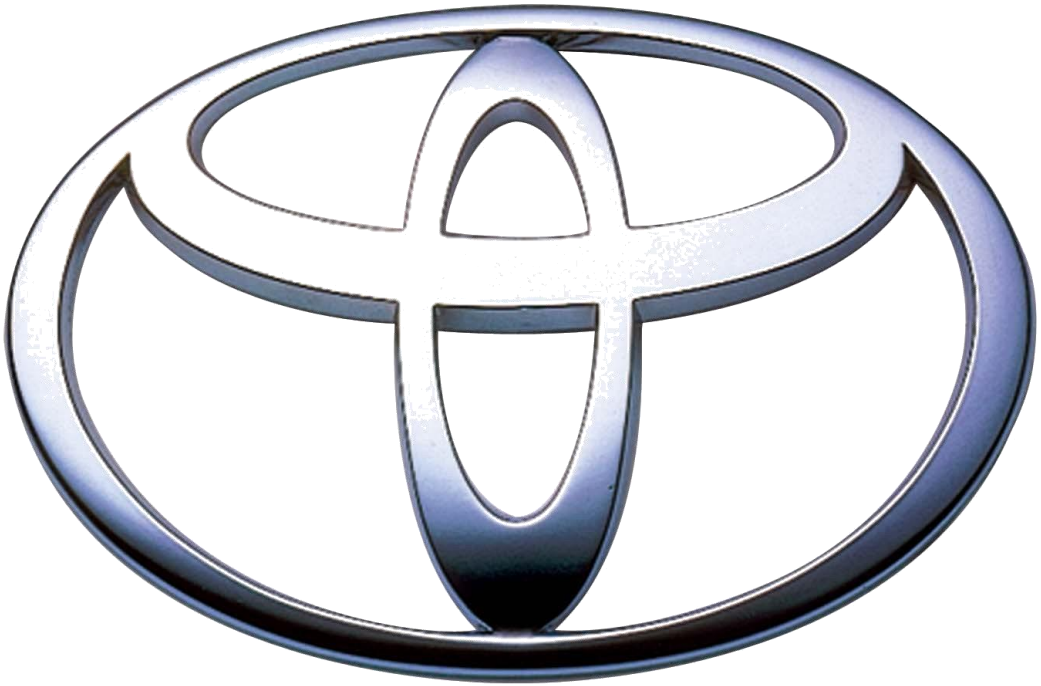


An example or requirement specification



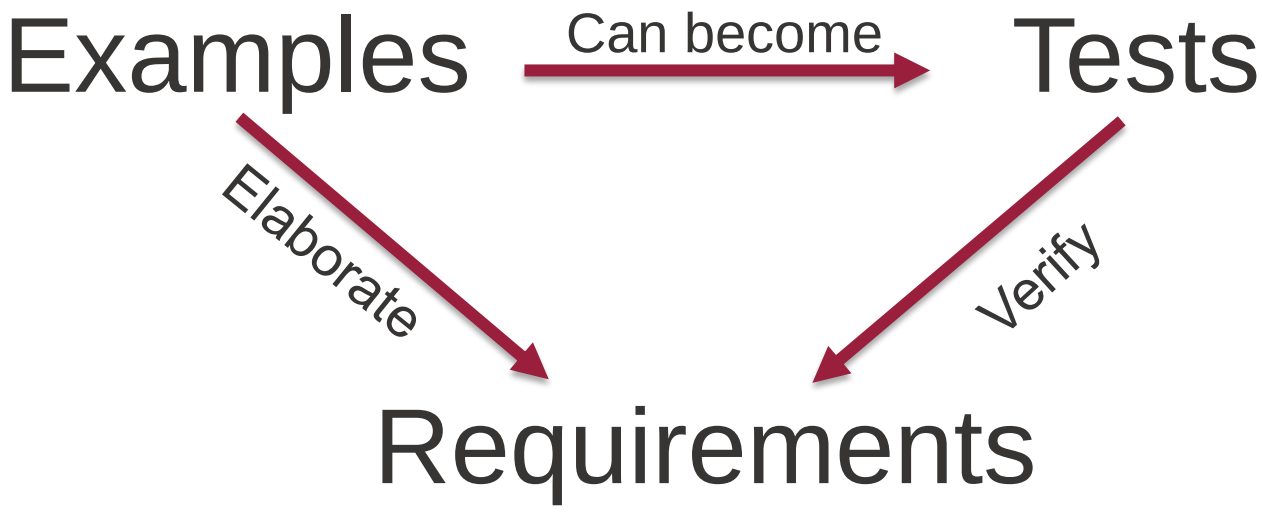
The Toyota Way

1. Check at the source
2. Verifications are (and should be) inexpensive
3. Test to prevent defects, not to discover them



CGI

The basics



Source: Gojko Adzic: Specification by Example

Generic approach of ATDD

- Use real-world examples to build understanding
- Create acceptance criteria from these examples
- Use acceptance criteria as specifications
- Create automated acceptance test cases
- Test delivered software using automated acceptance test cases

CGI

A good acceptance test is

- Focused on a single element (step, rule…)
- Not a script
- Self-explanatory
- SMART
  - Specific
  - Measurable
  - Achievable
  - Realistic
  - Time-bound

Implementing Acceptance Criteria

Validate that a story has been developed with the functionality the Product Owner had in mind.

- Test cases fill in the details of the story.
- Write tests before programming – Test first, build next.
- Execute test at end of sprint as demo
- Test cases are delivered as system documentation.

USER STORY CHECKLIST

Title:	***Book a flight***	
ID:	***US12a***	
Scenario:	***As a member of the F&H club I can use Fresh points to pay for a flight so that I can travel for free***	
Estimation:	***13 points***	Priority: ***High***

CGI

This is an acceptance criterion

Using test conditions and test cases as design documentation

Business Rules:
C1 - .....
C2 - **The return date must be later than or equal to the arrival date**
C3 - .....
C4 - .....

From Test Conditions to Test Cases

Test Cases in standard keyword driven format (TestFrame)

Test Condition	US12a-C2	The return date must be later than or equal to the arrival date				
Test case	US12a-C2-T2	Arrival date is later than return date				
	Login to F&H					
	Fresh & Honest Club number	*Password*				
login f&h club	**123.456.789.0**	**abcd**				
	Search a flight with invalid dates					
	Departure City	*Destination City*	*Departing*	*Returntrip*	*Returning*	
search flight	**ams**	**cdg**	**&Date(),17**	**Yes**	**&Date(),15**	
	Message					
check message	**Date of return flight is incorrect!**					

CGI

And this is a test case in another format

Given user "123.456.789.0" with password "abcd" is logged in
And departure city is "AMS"
And destination city is "CDG"
And return trip
When departure date is 17 days after today
And return date is 15 days after today
And query is submitted
Then the alert "Date of return flight is incorrect!" is shown

http://www.caplin.com/developer/component/verifier/reference/verifier-given-when-then-syntax-reference

Defining Acceptance Test Cases

How to define ATDD Test cases

Gherkin is a Business Readable, Domain Specific Language created especially for behavior descriptions. It gives you the ability to remove logic details from behavior tests.

Gherkin serves two purposes: serving as your project's documentation and automated tests. That also has a bonus feature: it talks back to you using real, human language telling you what code you should write.

CGI

Examples in Gherkin

In fact it is keyword driven testing

Given user "123.456.789.0" with password "abcd" is logged in
And departure city is "AMS"
And destination city is "CDG"
And return trip = “Y”
When departure date is 17 days after today
And return date is 15 days after today
And query is submitted
Then the alert "Date of return flight is incorrect!" is shown

Examples in Gherkin

In fact it is keyword driven testing

Keyword

Parameter

Given user "123.456.789.0" with password "abcd" is logged in
And departure city is "AMS"
And destination city is "CDG"
And return trip = “Y”
When departure date is 17 days after today
And return date is 15 days after today
And query is submitted
Then the alert "Date of return flight is incorrect!" is shown

CGI

Reuse using tables

Data driven testing

user	password	Dept city	Dest city	return	Dept date (now+#days)	Ret Date (now+#days)	Allert
123.456.789.0	abcd	AMS	CDG	N	13	-	Bookingdate must be 14 days in the future
123.456.789.0	abcd	AMS	CDG	Y	17	17	No alert

Given user "123.456.789.0" with password "abcd" is logged in

And departure city is "AMS"

And destination city is "CDG"

And return trip “Y”

When departure date is 17 days after today

And return date is 15 days after today

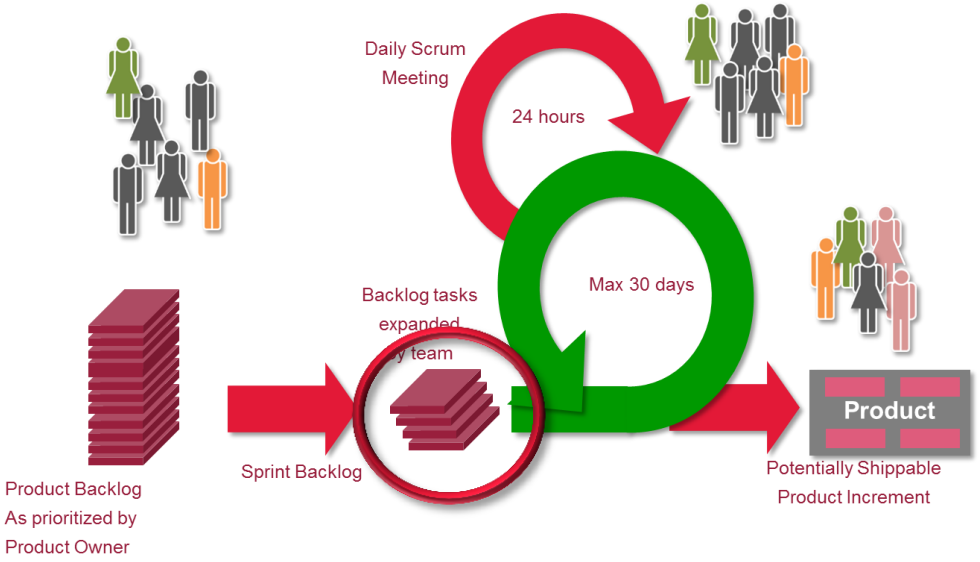
And query is submitted

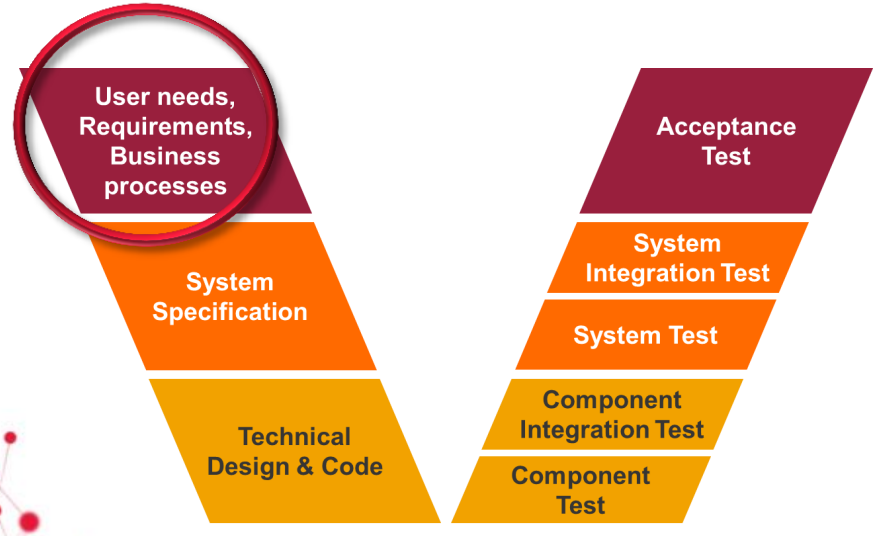
Then the alert "Date of return flight is incorrect!" is shown

CGI

The agile paradigm shift

Due to agile development we achieved what seemed to be unachievable in V-model development:





As a tester being involved at the very start of systems development!

CGI

The workshop
Creating test cases

Experience the commitment®

Tooling

Experience the commitment®

Tools available

Jbehave:	http://jbehave.org/
Fit:	http://fit.c2.com/
FitNesse:	http://fitnesse.org/
Easyb:	http://www.easyb.org/
Cucumber:	http://cukes.info
Robot:	http://code.google.com/p/robotframewo
Arbiter:	http://arbiter.sourceforge.net/
Concordian:	http://www.concordion.org/
Selenium:	http://seleniumhq.org
Watir:	http://watir.com/
Twist:	http://www.thoughtworks.com/products/twist-agile-testing

Demo Robot Framework

CGI

Experience the commitment®

More information: some books

Lisa Crispin et al.:	Agile Testing: A Practical Guide for Testers and Agile Teams
Kent Beck:	Test Driven Development: By Example
Kent Beck:	Extreme Programming Explained: Embrace Change,
Markus Gärtner:	ATDD by Example: A Practical Guide to Acceptance Test-Driven Development
Ken Pugh:	Lean-Agile Acceptance Test-Driven Development
Gojko Adzic:	Specification by Example