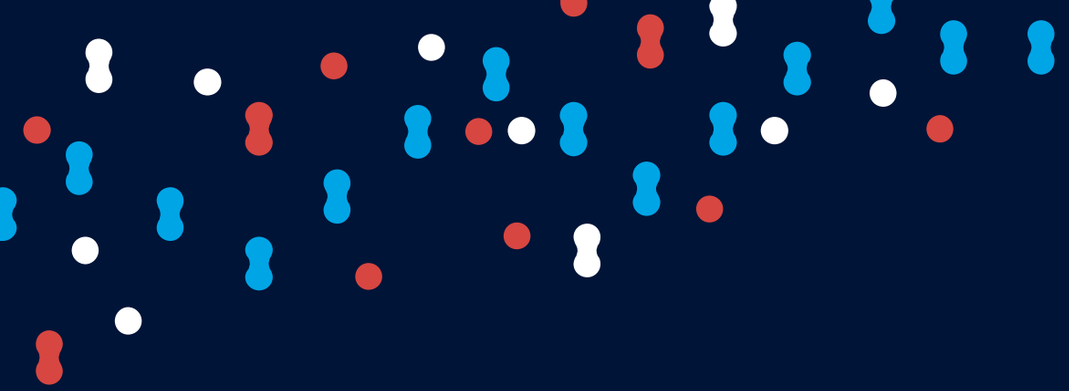


Testing Automation and Methodologies

14 May 2020

Agenda

Specification by example & ATDD

- From Requirements to User Stories
- Specification by Example
- Acceptance Test Driven Development

Writing tests in ION

- Testing against asynchronous systems
- Kinds of validations

Robot Framework

- What is
- Architecture & libraries
- Testing styles and formats
- Reports

Continuous Integration and Testing Automation

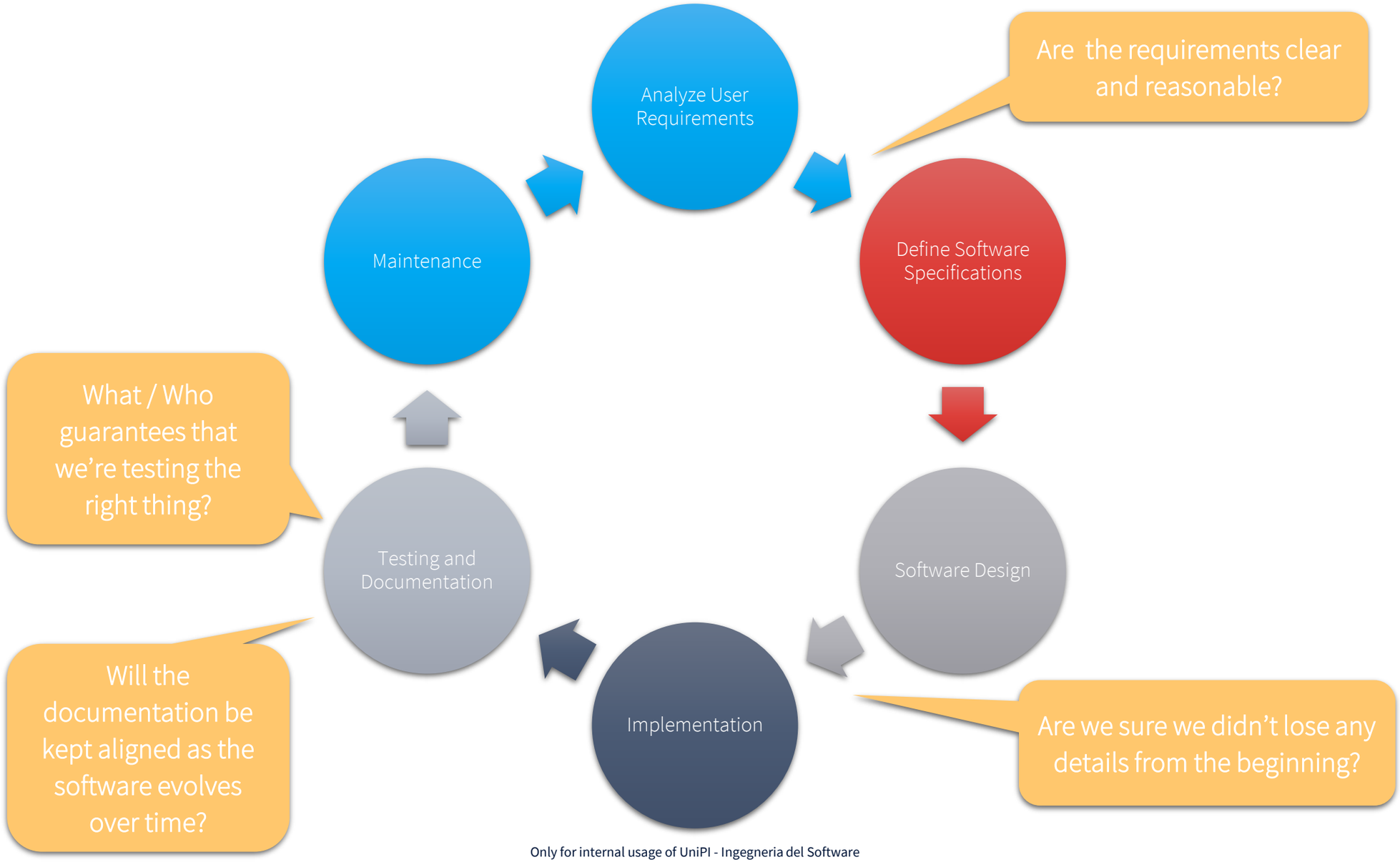
- Problems that we want to address
- What is Continuous Integration
- Testing Automation with Jenkins and Robot Framework

Specification by example

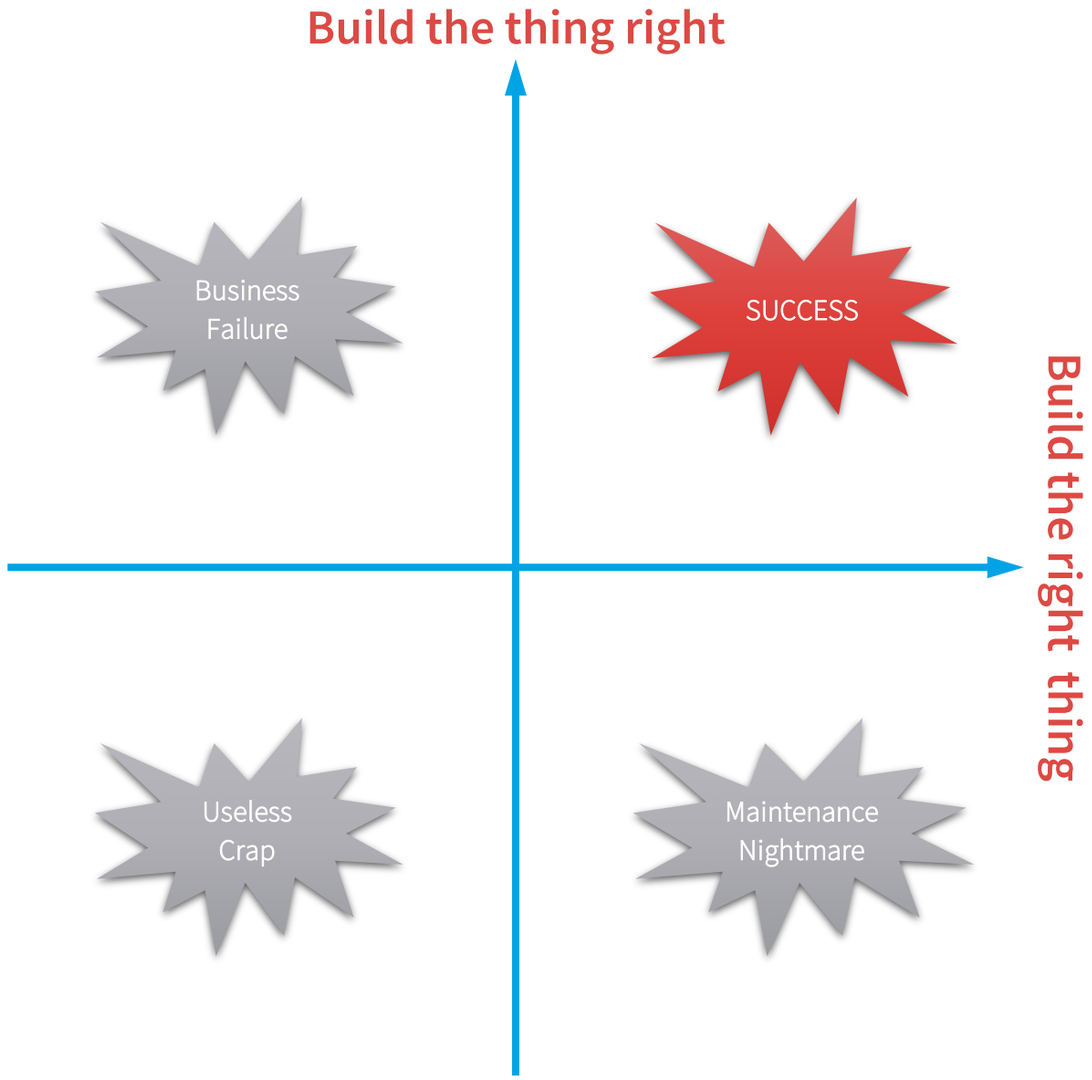
&

ATDD

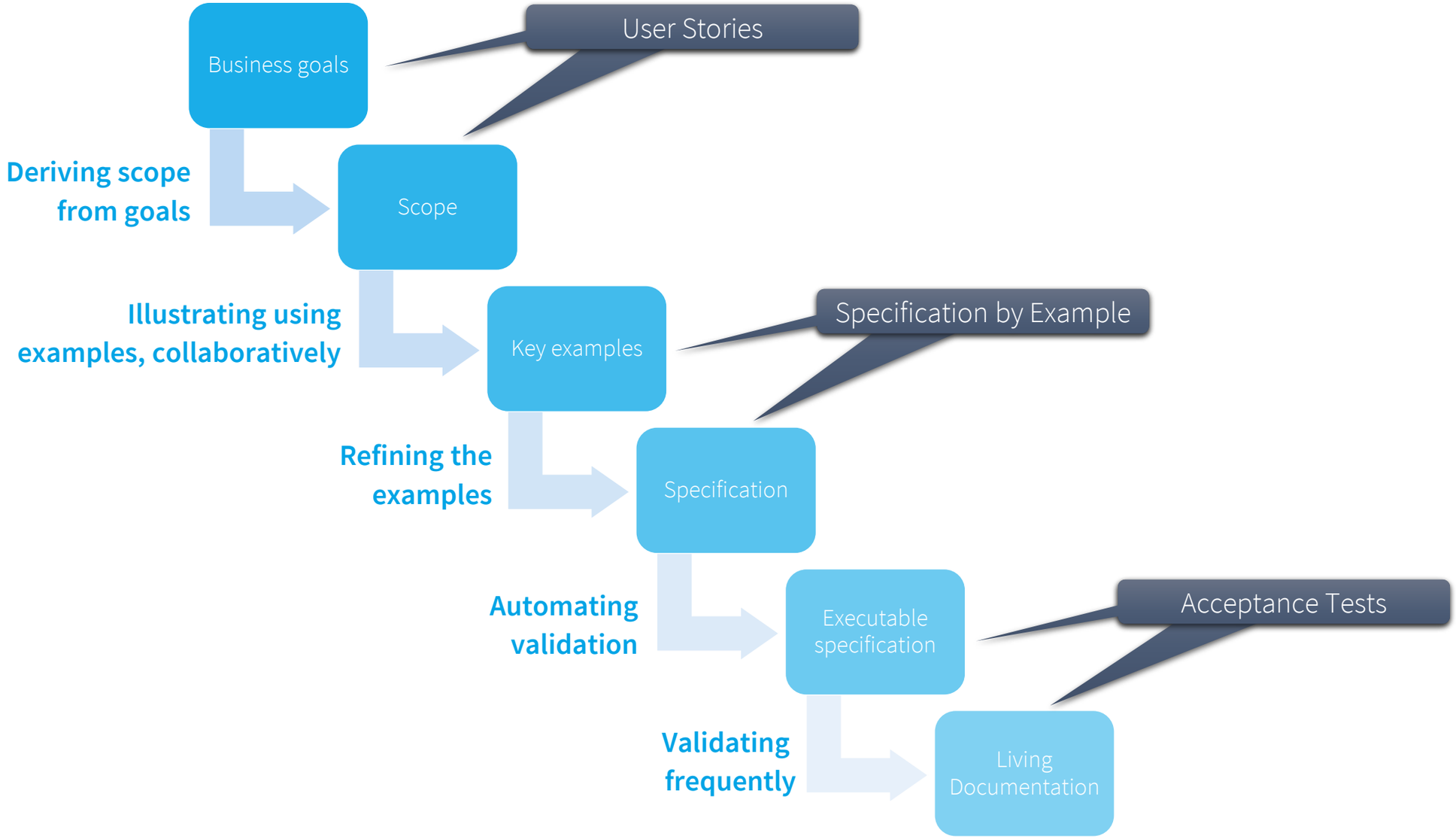
Software Development Life cycle (classic)



Software Development – Possible outcomes



Spec by example & ATDD
Reviewed development workflow



Deriving the scope from the goals

F-16 design team was asked to do the impossible: a cheap 2.5 Mach airplane!

When asked why they need Mach 2 - 2.5, the answer was *“to be able to escape from combat.”*

The solution was **providing acceleration and maneuverability**, not maximum speed.

Refuse requirements that are a solution to an unknown problem!

Defining the scope

User Stories

A user story is an informal, natural language description of one or more features of a software system.

- Writing a user story is a good starting point to identify the business goals and their scope
- A user story must specify the **actor** and the **reason** of a **feature**

"As a …… I want …….. in order to ……."

As a Risk Manager, I want to limit the quantity traded by junior traders in order to avoid an excessive exposure.

Illustrating using examples, collaboratively

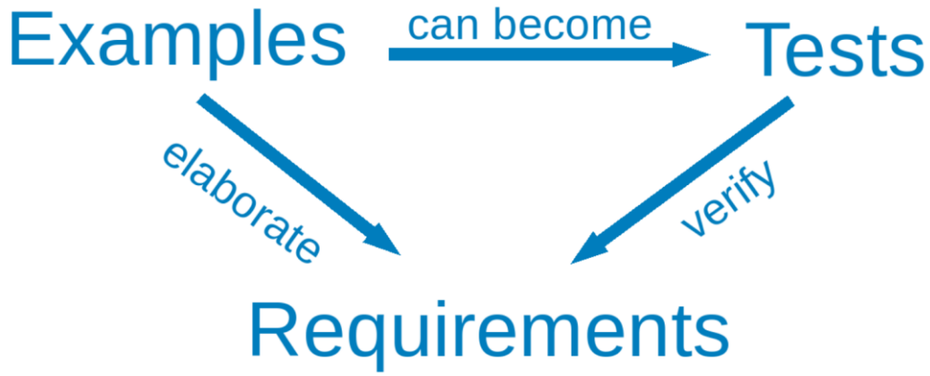
Evaluate how long it takes to understand the following:

- Use a yellow circle
- Divide in two sections, top and bottom
- In the top section, there are two black filled circles, equally distant from the center
- In the bottom section, there is a curved arc equally distant from the outer circle, with two smaller arcs at the end

Illustrating using examples, collaboratively

Specification by example

Set of collaborative process patterns that facilitate changes in software products, to ensure that the right product is delivered effectively.



Specification by example

- Use simple, concrete and concise examples
- Use real data and avoid abstract descriptions
- Examples must be easy to understand
  - They should trigger a discussion in the team!

User Story

As a risk manager, I want to limit the quantity traded by junior traders in order to avoid an excessive exposure.

Examples

RM: “Jeff is a junior trader, he should not create order with a quantity greater than 10k”.

RM: “Paul instead is a senior trader, he’d be allowed to do it”.

Specification by example

The specification by example is the result of a discussion among the team members

- Together with the stakeholders, the team members distill some key examples
- Additional examples can be added if considered relevant for the feature
- Standard formats to express the samples ease the process of making them *executable* (e.g. tabular format)

Trader	Order Quantity	Outcome
Junior	15 K	REJECTED
Senior	15 K	ACCEPTED
Junior	10 K	ACCEPTED

This table can easily become a true Acceptance Test!

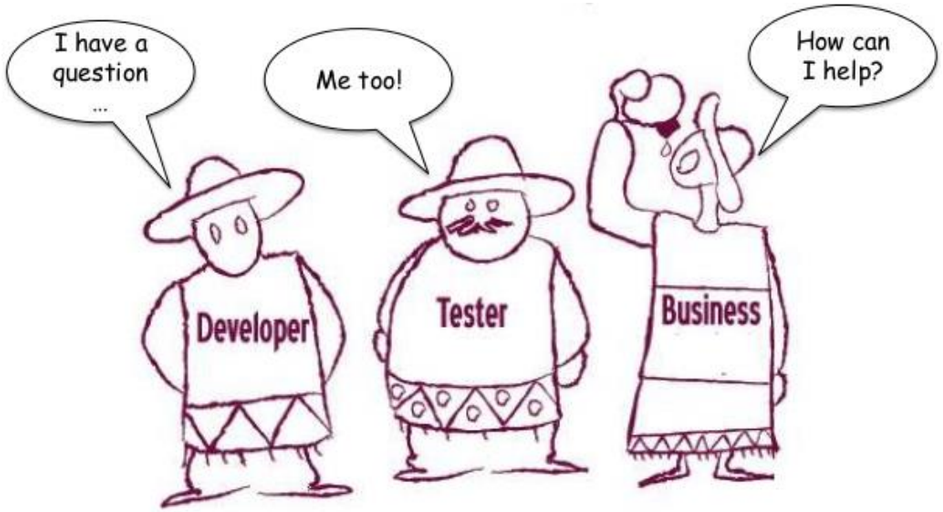
Collaboration model

Specifications (and tests) should be produced by a team composed of different skills.

Possible models are:

- Large workshop, such as a PBR
- Three amigos
  - Business Analyst
  - Developer
  - Tester
- Quick discussion, offline reviews

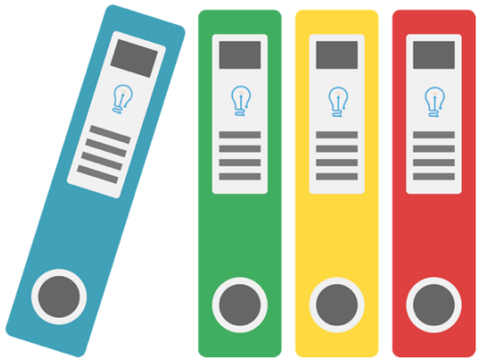
- Note: developers and testers are often the same people!



Make the examples executable documentation

- A non-trivial system needs some level of documentation to be maintained and used properly
- Usual paper documentation can become old pretty soon, thus unable to properly describe the system behaviors

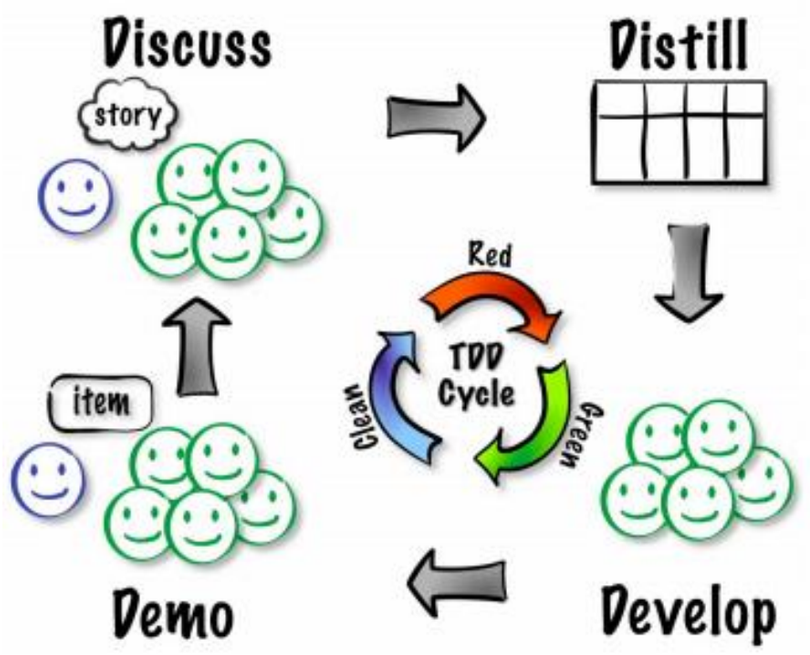
An authoritative source of truth for everybody is required



Acceptance Test Driven Development

ATDD

Software development process that involves team members with different perspectives
(customer, development, testing)
collaborating to write Acceptance Tests in advance of implementing the corresponding functionality.



What is an Acceptance Test?

An Acceptance Test is a procedure aimed to validate a feature respecting to the requirements

Acceptance Tests:

- can be easily derived from the specifications by example
- describe a specific behavior of the system (they are the effective documentation of a feature)
- are a live specification between the Dev Team and the Product Owner
- are readable and writable also by non-technical people

The expectations contain only details relevant for the feature

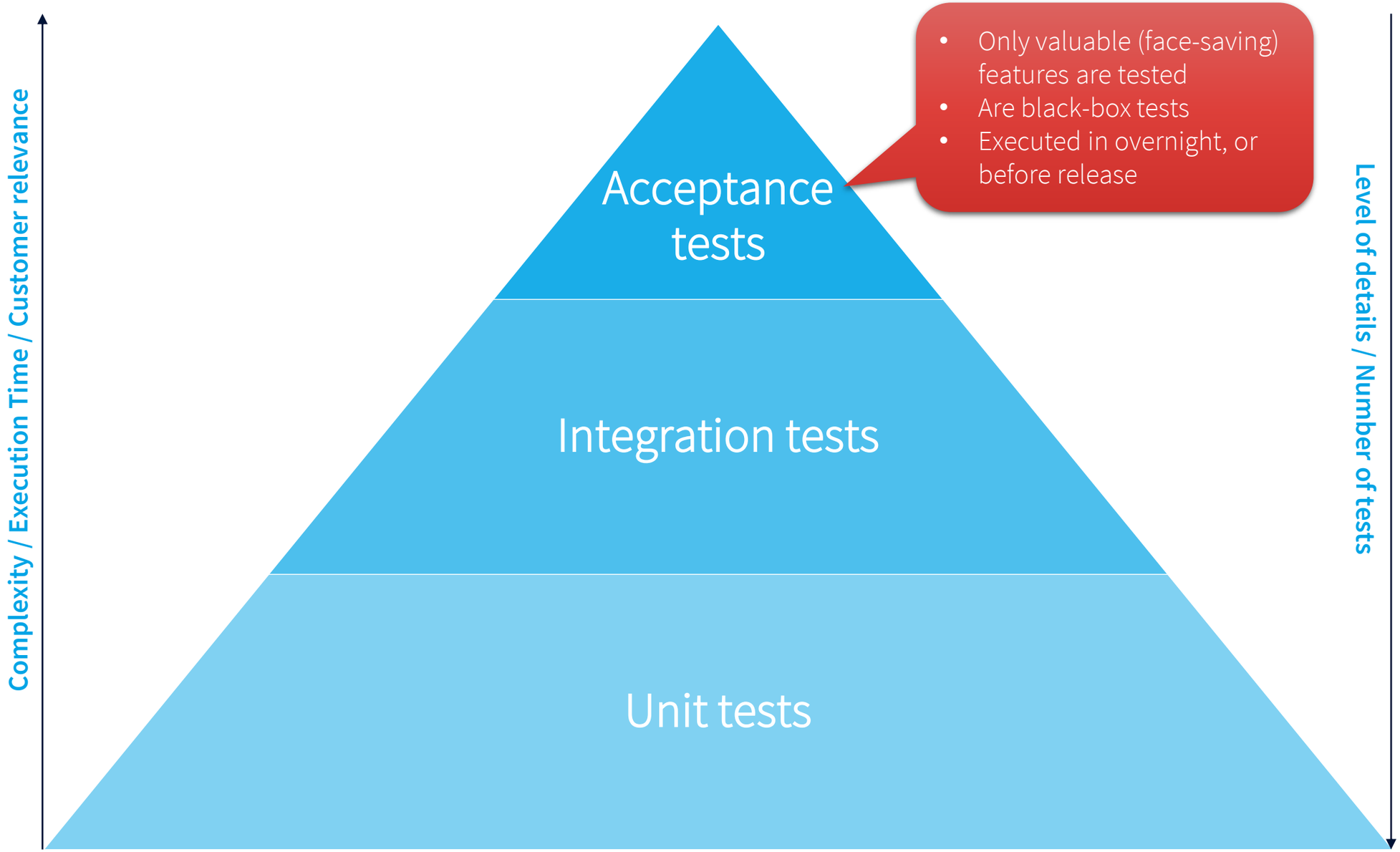
#	Price	Quantity	Verb	Outcome	ExpPrice	ExpQuantity	ExpVerb
Adding a valid Trade	100	10	BUY	TRADE BOOKED	100	10	BUY
Adding an invalid Trade because Quantity is negative.	100	-10	BUY	FAILURE	-	-	-

Only relevant cases are tested

A testing framework can parse this table and execute the test. Example: **Robot Framework**.

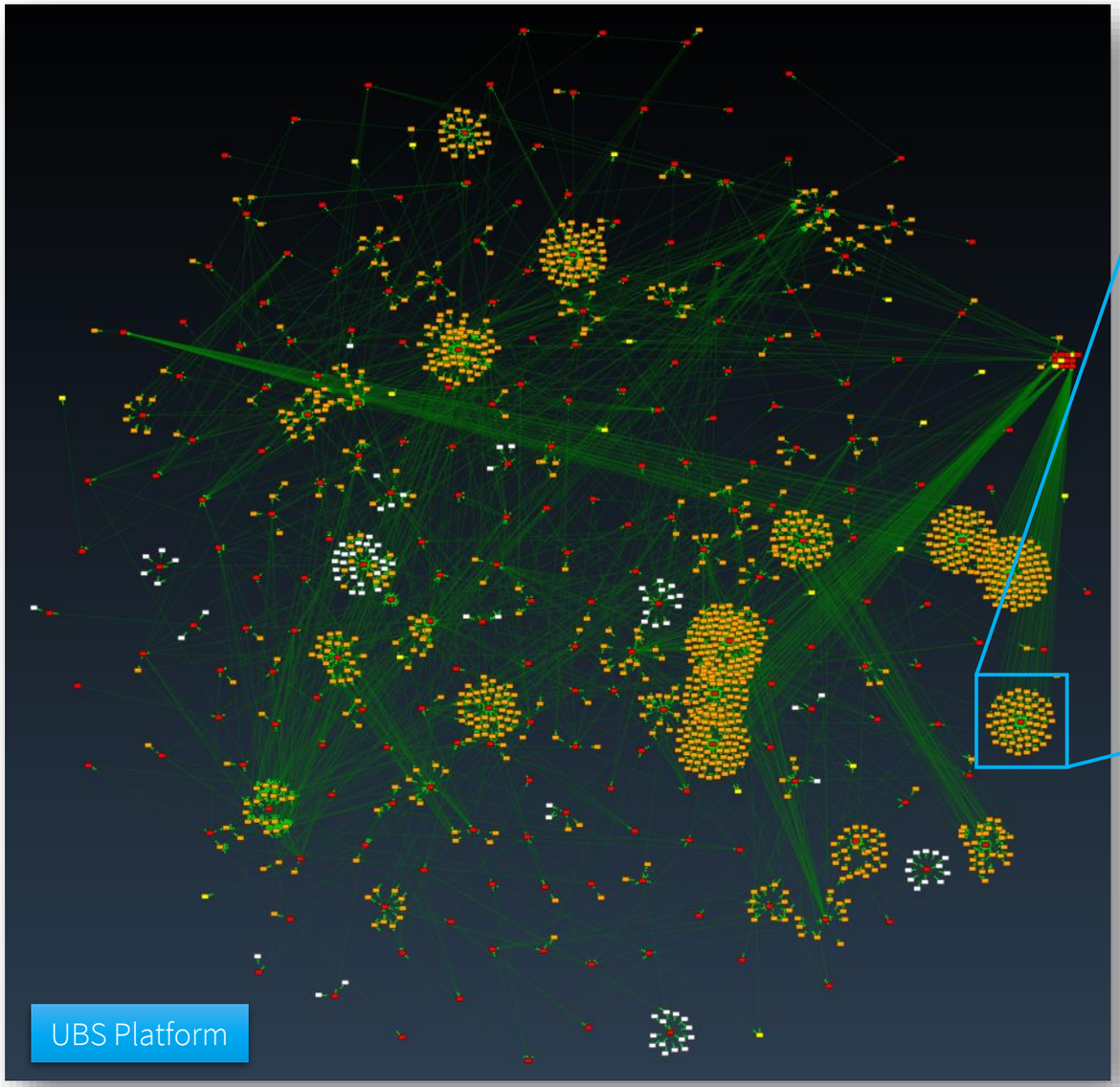
Tabular format: the one preferred in ION

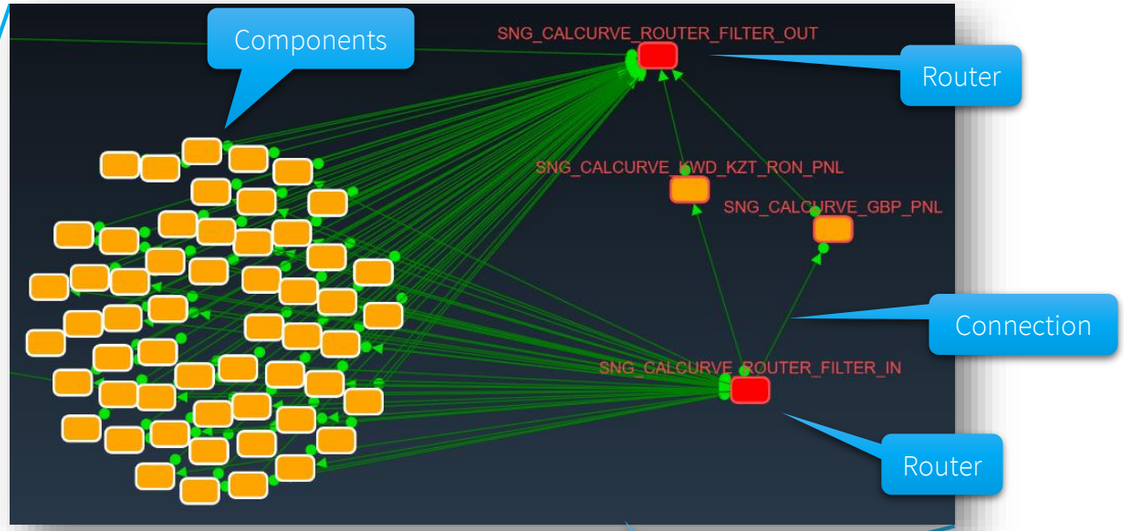
Pyramid of tests



Writing tests in ION

Writing tests in ION – The ION Platform





Communication patterns

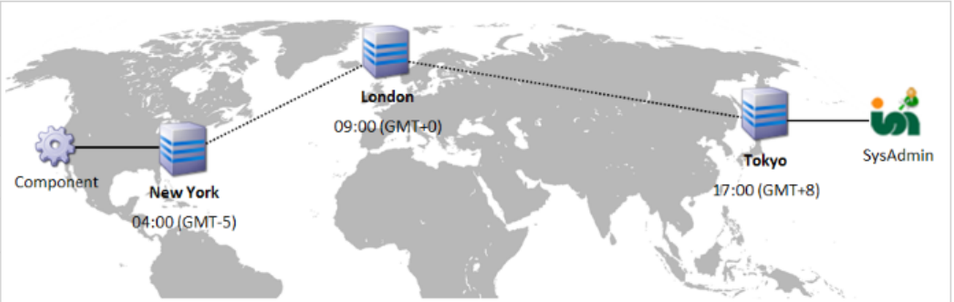
- Publish \ Subscribe
- Request \ Reply
- Message Queues

Special components, called Daemons, administer the whole Platform.

Spec by example & ATDD
ION example of Acceptance Test

LOCATION BASED COMPONENT SCHEDULING

Settings		
Resource	..${/}..${/}init.html	
Force Tags	location_based_comp_scheduling	112
Suite Teardown	**Set Registration Daemon**	**MASTER**



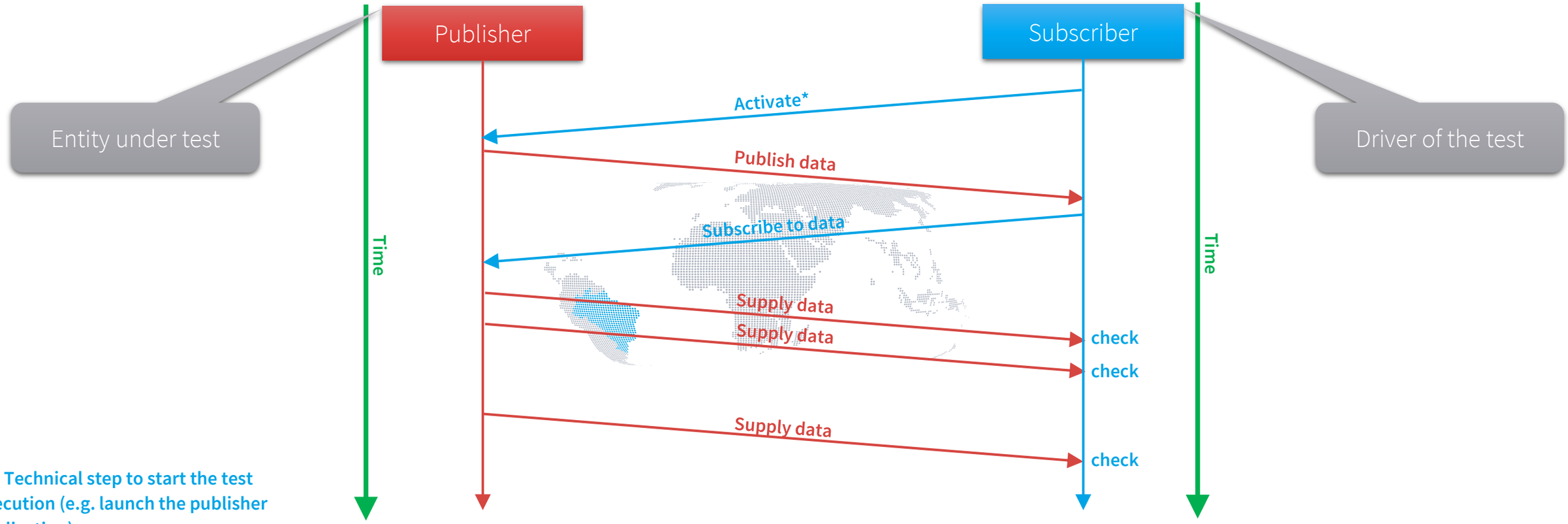
Test Cases

Location Based Component Scheduling Test Case					[Template] **Location Based Component Scheduling**
The test will generate a schedule like "hhmm-hhmm" using the Component Time Zone, apply it to the Time Zone's week day, and verify that the component is scheduled accordingly.					
Component Registration Daemon	**Component Time Zone**	**Component Status**	**Scheduled START (Minutes from now)**	**Scheduled STOP (Minutes from now)**	**Expected Output**
Default / Legacy Behavior					
MASTER	Local	Waiting	-1	1	Component started immediately and stopped in 1 minute.
MASTER	Local	Waiting	-2	-1	Nothing to do today.
MASTER	Local	Running	1	2	Component stopped immediately, started in 1 minute, and stopped in 2 minutes.
MASTER	Local	Running	-1	1	Component stopped in 1 minute.
MASTER	Local	Running	-2	-1	Component stopped immediately.
Time Zones with negative offsets					
MASTER	(UTC-12:00) International Date Line West	Waiting	-1	1	Component started immediately and stopped in 1 minute, using the (UTC-12:00) International Date Line West Time Zone.
MASTER	(UTC-06:00) Central Time (US & Canada)	Waiting	1	2	Component started in 1 minute and stopped in 2 minutes, using the (UTC-04:00) Caracas Time Zone.
MASTER	(UTC-05:00) Eastern Time (US & Canada)	Waiting	-1	1	Component started immediately and stopped in 1 minute, using the (UTC-01:00) Cabo Verde Is. Time Zone.
Time Zones with positive offsets					
MASTER	(UTC+01:00) Amsterdam, Berlin, Bern, Rome, Stockholm, Vienna	Waiting	1	2	Component started in 1 minute and stopped in 2 minutes, using the (UTC+01:00) Sarajevo, Skopje, Warsaw, Zagreb Time Zone.
MASTER	(UTC+01:00) Brussels, Copenhagen, Madrid, Paris	Waiting	-1	1	Component started in 1 minute and stopped in 2 minutes, using the (UTC+05:30) Sri Jayawardenepura Time Zone.

Writing tests in ION

Under the hood, in ION it's always a matter of stimulating an asynchronous system and waiting for a feedback

- Example: processes interacting using the publish \ subscribe over a network connection
- When the system is perturbed, you'll possibly get a feedback after a certain amount of time

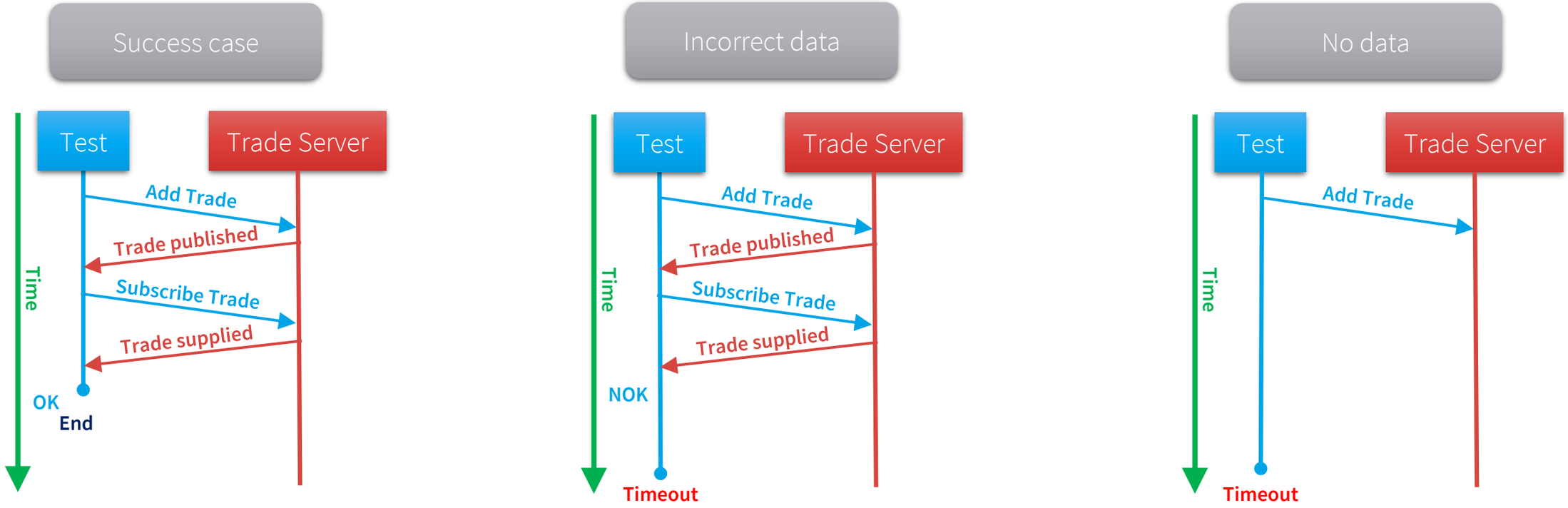


(*): Technical step to start the test execution (e.g. launch the publisher application).

Writing tests in ION

Example

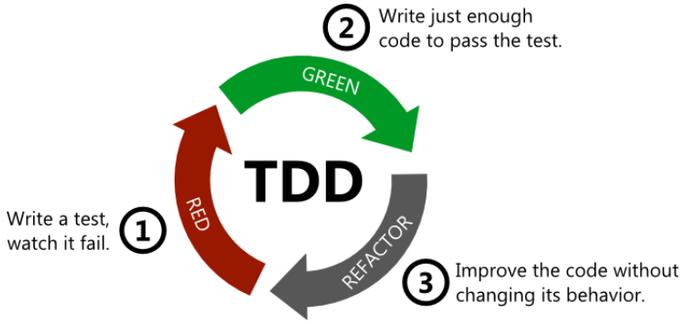
- Suppose a Trade Server component that, when a trade is made, should publish a record containing the trade data.
  - We want to verify the creation of the trade record with the expected data.
  - As soon as we see the condition satisfied, we can proceed further in our test.



Writing tests in ION

Writing tests against asynchronous environments

- Testing strategies should be carefully chosen
  - to avoid writing complicated code to interact with the environment (maintainability)
  - to avoid the usage of anti-patterns
    - Sleeps (let me sleep for enough time before making the check)
    - Polls (repeat until passes {check!})
- The timeout is a first class citizen
  - e.g.: the halting problem, some real cases:
    - Infinite loops
    - Deadlocks
    - Crashes
- Choosing the right timeout to be used is critical
  - Too short -> possible premature failures -> the test is unstable
  - Too long -> waste of time, but only in case of unmet expectations..
    - Quite annoying, expecially when adopting the Test Driven Development



Robot Framework

Introduction

What is it?

Robot Framework is a generic open source test automation framework for acceptance testing and acceptance test-driven development (ATDD).

Who does maintain it?

Initially developed by Pekka Klarck at **Nokia Networks** in 2005, it is now maintained by the **Robot Framework Foundation**, a non-profit consortium with the focus on developing, fixing bugs and managing community requests for Robot Framework.

Introduction

Which license?

Robot Framework is released under the Apache License 2.0.

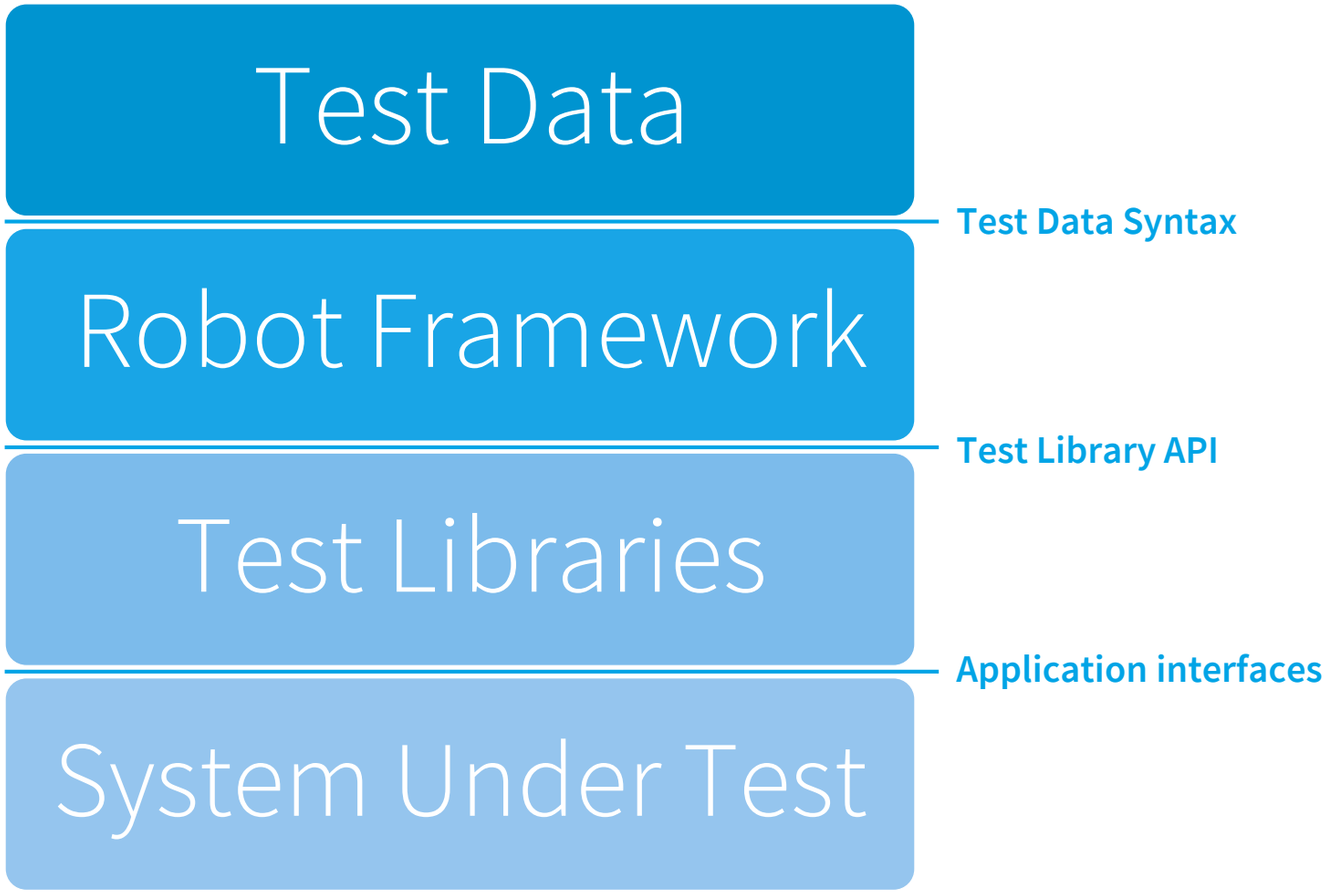
What kind of technology?

The core framework was implemented using **Python**, but runs also in Jython (JVM) and IronPython (.NET).

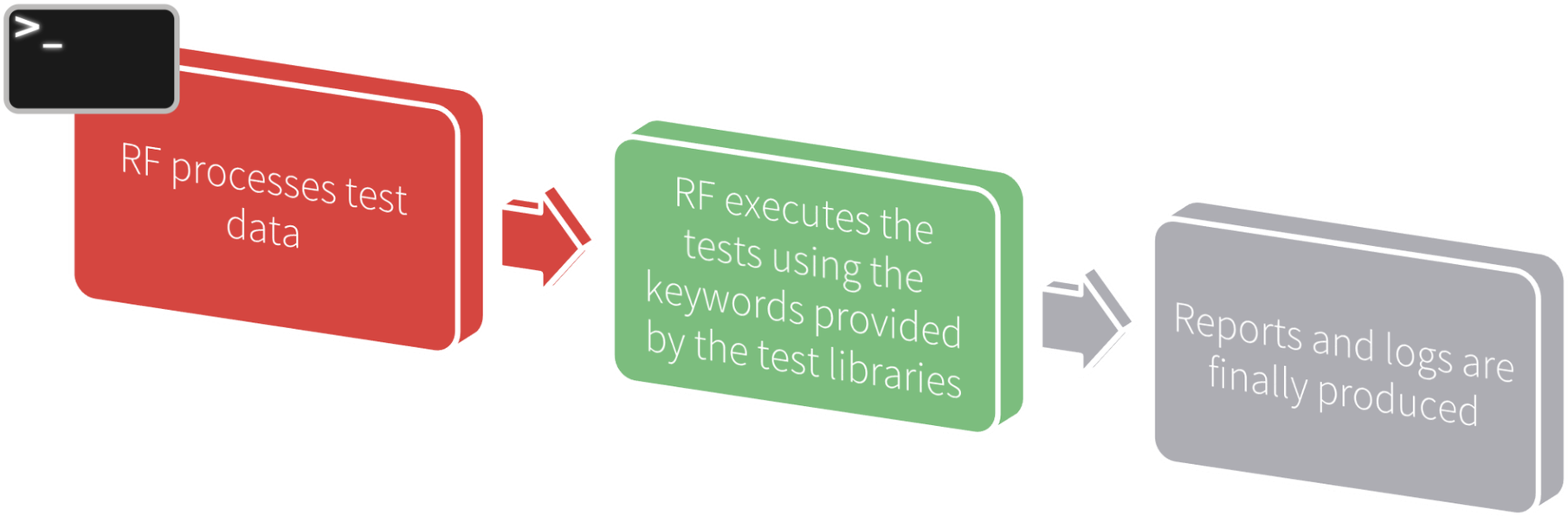
Why?

- Native support to different testing syntaxes
- Modular system, it can be extended through test libraries
- Easy to read reports and logs in HTML format

Architecture



Workflow



Robot Framework does not know anything about the system under test.

The test libraries act as a driver between Robot Framework and the system under test.

Robot Framework
Standard Testing Libraries

Libraries are officially maintained and distributed by the Robot Framework foundation

BuiltIn	Collections	DateTime	Operating System
Generic kewyords, always imported	To handle lists and collections	To handle date and time values	Basic OS activity, such as copy file

Process	Strings	XML	Telnet
Support for process execution	Manipulations of strings	To verify and modify XML documents	Telnet connection and commands over connection

Robot Framework
External Testing Libraries

External libraries maintained by the community

STANDARD	EXTERNAL	OTHER
Android library Library for all your Android automation needs. It uses Calabash Android internally.	**AnywhereLibrary** Library for testing Single-Page Apps (SPA). Uses Selenium Webdriver and Appium internally.	**AppiumLibrary** Library for Android- and iOS-testing. It uses Appium internally.
Archive library Library for handling zip- and tar-archives.	**AutoItLibrary** Windows GUI testing library that uses AutoIt freeware tool as a driver.	**CncLibrary** Library for driving a CNC milling machine.
ConfluentKafkaLibrary Library for python confluent kafka.	**CURFLibrary** Library for testing CAN bus with support for ISO-TP and UDS.	**Database Library (Java)** Java-based library for database testing. Usable with Jython. Available also at **Maven central**.
Database Library (Python) Python based library for database testing. Works with any Python interpreter, including Jython.	**DataDriver Library** Library for Data-Driven Testing with external data tables (csv, xls, xlsx, etc.). Pairwise Combinatorial Testing support.	**Debug Library** A debug library for RobotFramework, which can be used as an interactive shell(REPL) also.
Diff Library Library to diff two files together.	**Django Library** Library for **Django**, a Python web framework.	**Eclipse Library** Library for testing Eclipse RCP applications using SWT widgets.
robotframework-faker Library for **Faker**, a fake test data generator.	**FTP library** Library for testing and using FTP server with Robot Framework.	**HTTP library (livetest)** Library for HTTP level testing using livetest tool internally.
HTTP library (Requests) Library for HTTP level testing using Request internally.	**HttpRequestLibrary (Java)** Library for HTTP level testing using Apache HTTP client. Available also at **Maven central**.	**iOS library** Library for all your iOS automation needs. It uses Calabash iOS Server internally.
ImageHorizonLibrary Cross-platform, pure Python library for GUI automation based on image recognition.	**JavaFXLibrary** Library for testing JavaFX applications, based on **TestFX**. Has also remote interface support.	**KiCadLibrary** Library for interacting with KiCad EDA designs.
MongoDB library Library for interacting with MongoDB using pymongo.	**Mainframe3270 Library** Library that allows to create automated test scripts to test IBM Mainframe 3270.	**MQTT library** Library for testing MQTT brokers and applications.
NcclientLibrary https://github.com/ncclient/ncclient	**Rammbock** Generic network protocol test library that offers easy way to specify network packets and inspect the results of sent and received packets.	**RemoteSwingLibrary** Library for testing and connecting to a java process and using SwingLibrary, especially Java Web Start applications.
RESTinstance Robot Framework test library for HTTP JSON APIs.	**RPA framework** Collection of open-source libraries and tools for Robotic Process Automation (RPA), designed to be used both with Robot Framework and Python.	**SapGuiLibrary** Library for testing the SAPGUI client using the internal SAP Scripting Engine
Selenium2Screenshots Library for capturing annotated screenshots with Selenium2Library.	**SeleniumLibrary** Web testing library that uses popular Selenium tool internally.	**SeleniumLibrary for Java** Java port of the SeleniumLibrary.
SikuliLibrary Sikuli Robot Framework Library provide keywords to test UI through **Sikulix**. This library supports Python 2.x and 3.x.	**SSHLibrary** Enables executing commands on remote machines over an SSH connection. Also supports transfering files using SFTP.	**SudsLibrary** A library for functional testing of SOAP-based web services based on Suds, a dynamic SOAP 1.1 client.
SwingLibrary Library for testing Java applications with Swing GUI.	**TestFX Library** Library to enable to test Java FX applications using the **TestFX framework**. Has also remote interface support.	**TFTPLibrary** Library for interacting over **Trivial File Transfer Portocol**.
WhiteLibrary Library for automating Windows GUI. Based on White framework, supported technologies include Win32, WinForms, and WPF.	**watir-robot** Web testing library that uses Watir tool.	

Key concepts

Test case

- Single unit of testing
- Usually defined from a Specification by Example (SBE), covers a user story

Test suite

- Collection of test cases
- Every file containing a test case is considered a test suite
- Folders are considered test suites

Keyword

- Single unit of execution
- Keyword are the basic building blocks of test cases
- Keywords can be composed together to create higher-level keywords

Library

- Collection of keywords

Supported test data formats: HTML

Sections are identified by the text contained in the first cell of a table

Test Cases		
My first test	*[Documentation]*	This is a very simple test
	Log	This is a log message
	Send Trade With Qty	100
My second test	*[Documentation]*	This is another very simple test
	Check DB table	INSTRUMENT

Table cells are used to naturally separate values

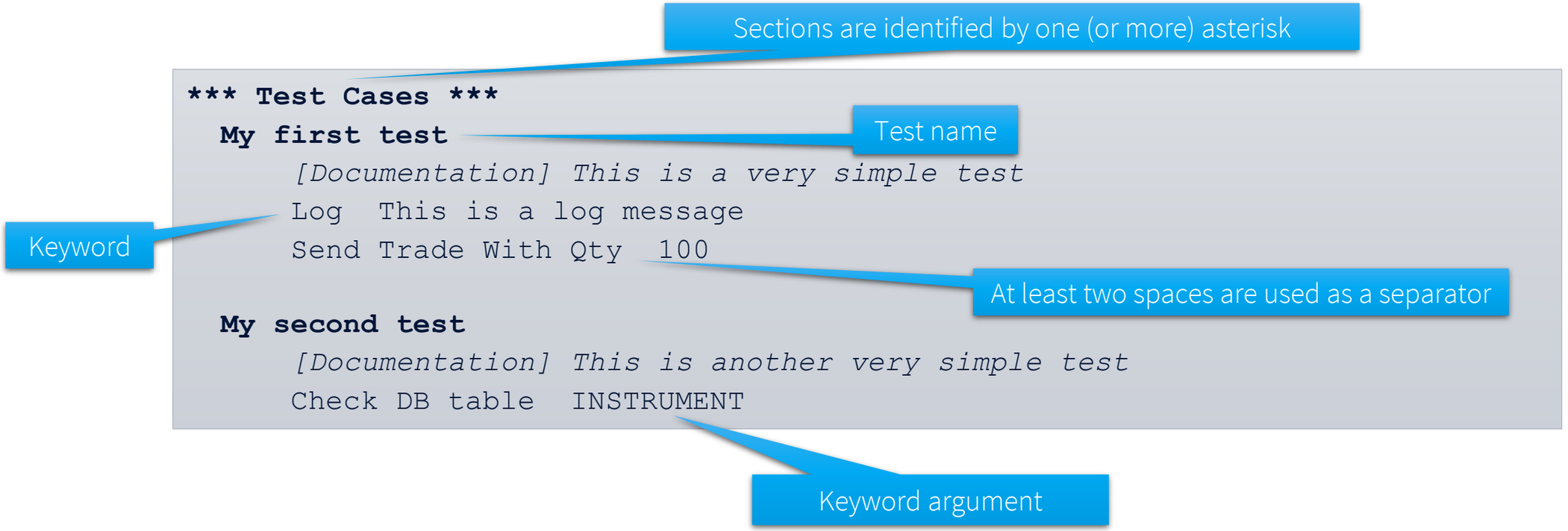
Pros

- Nice to read

Cons

- Requires HTML editor
- Not SCM-friendly

Supported test data formats: Plain text



Pros

- Easy to edit with any editor
- SCM friendly

Cons

- Readability not excellent

Testing styles

Keyword driven (it can be used to describe a flow)

Test Cases	Action	Argument	Argument
User cannot login with a bad pwd	Create Valid User	Fred	12345
	Login With Credentials	Fred	6789
	Status Should Be	Access Denied	

Behavior driven (a.k.a. Given-When-Then or Gherkin style, good fit for SBEs)

Test Cases	Action	Argument	Argument
User cannot login with a bad pwd	**Given** a Valid User	Fred	12345
	When He Logins With Credentials	Fred	6789
	Then The Status Should Be	Access Denied	

Given/When/Then terms are ignored

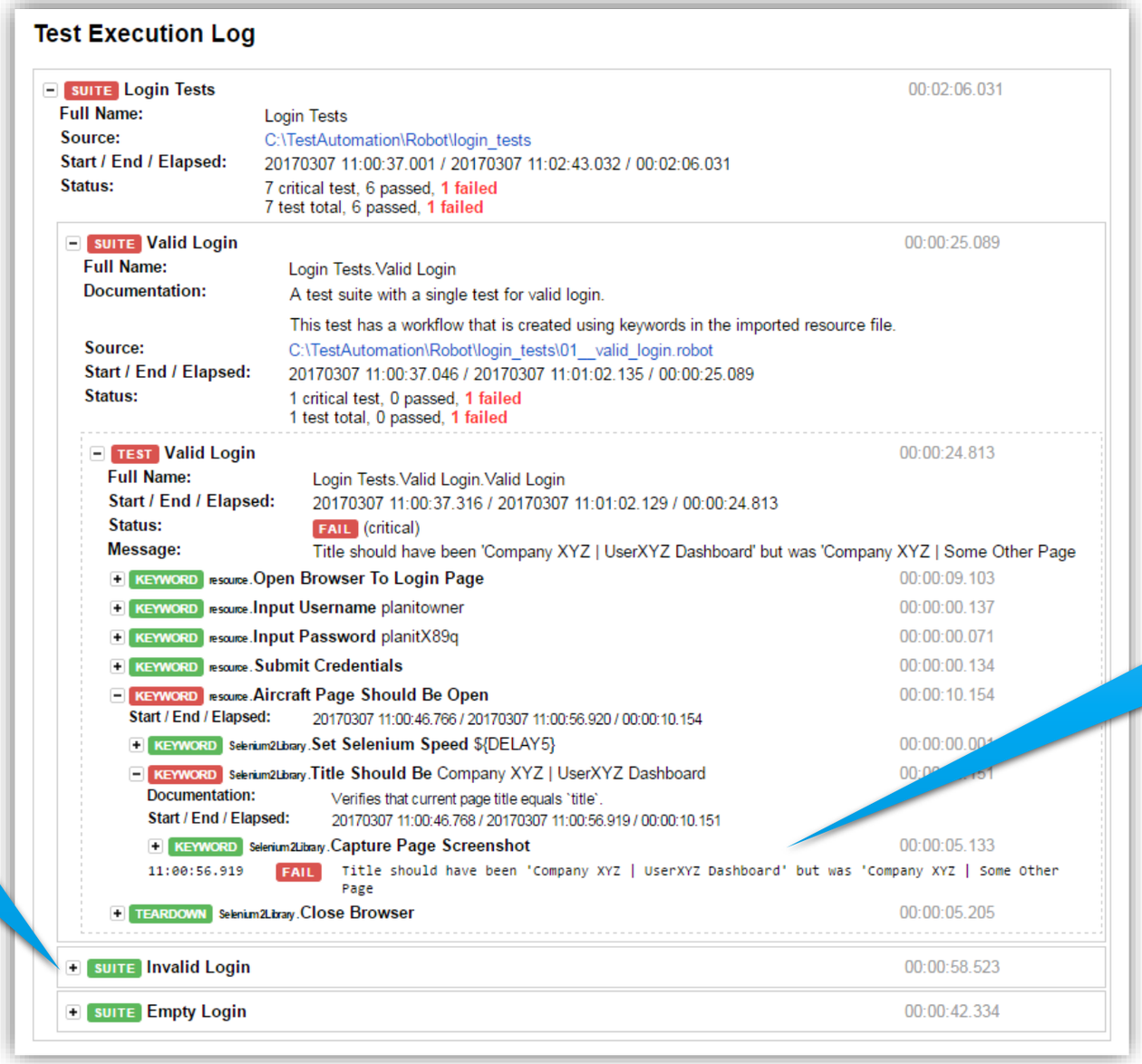
Data driven (another good fit for SBEs)

Input columns

Output columns

Test Cases			
User cannot login with a bad pwd	[Template]	Do login and verify status	
#	*Existing user*	*Login credentials*	*Expected status*
	Fred : 12345	Fred : 6789	Access denied
	Fred : 12345	Fred : 12345	Ok

Robot Framework
Reports



Passing tests are collapsed by default, allowing anyway manual drill down

Automatically drilled-down to the steps that caused the test failure

Creating a testing library

Robot Framework can be extended with a custom library of keywords to interact with a specific domain

Easiest and simplest way: set of user keywords in different files, loaded as resources

- Ok for simple keywords and as a first step
- Not recommended for more complex activities (Robot Framework is not a programming language)

More advanced way: keywords defined in Java, C#, or Python, using the APIs offered by Robot Framework

- Perfect choice to hide complex technical activities
- Proper developing effort is required (library code is like production code!)

ION has created its own Robot Framework library that allows to interact with the ION domain

The library is used both internally and by ION clients to perform end to end tests on the ION domain alone or integrated with third party systems.

Continuous Integration

What problems we want to address?

Typical integration scenario

- You created or changed the code of a project
- To make it usable, you need to integrate it with a bigger solution
- Integration here means building the whole solution: compile, package, test
- The build is the ultimate artifact of the integration process

When does it happen?

- Testing for QA
- Demo for business
- New release

What problems we want to address?

Typical issues

- Conflicts with code committed by other developers
- The project does not build
- The program does not work as expected
- The program does not work on some platforms (Operating System, Database, ...)

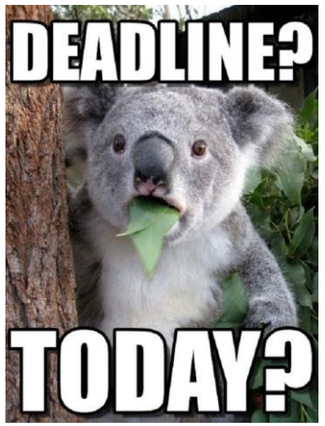
Issues grow together with the solution complexity

- Bigger dev teams, more coordination issues
- More supported platforms, more portability issues
- More modules working together, more integration issues

When integration goes wrong

And when the release deadline comes…

- "It doesn't compile!"
- "The library is old!"
- "Who did commit that #$%^&!?"
- "Sorry Dear, we have to release! I will come home tomorrow.."



How to avoid the typical integration issues

Key facts to avoid such problems

- Commit code frequently
- Deploy and test frequently
- Fix bugs promptly
- Make it a flow
- Automate
  - Build
  - Testing
  - Reporting

How can we achieve this?

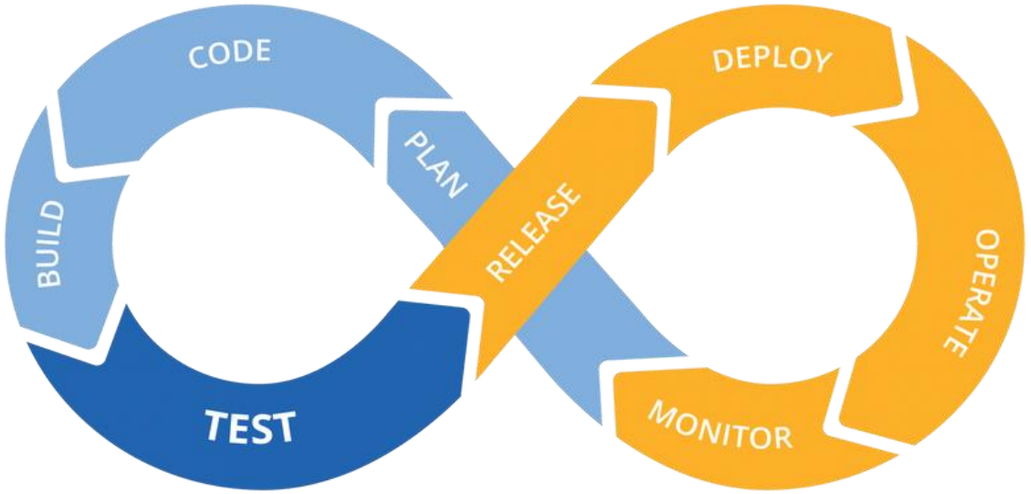
The answer is Continuous Integration

What is Continuous Integration?

Continuous Integration (CI)

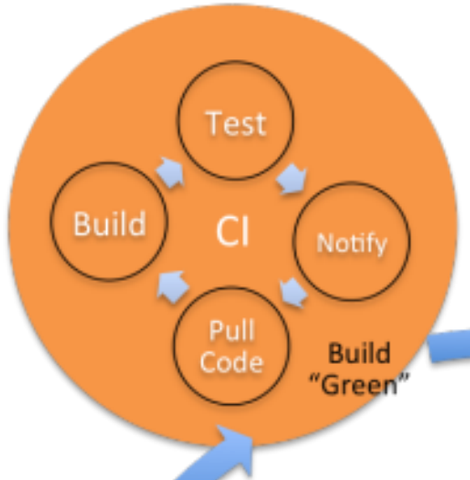
It’s a development practice that requires developers to integrate code into a shared repository several times a day.

- Each check-in is then verified by an automated build, allowing teams to detect problems early.
- By integrating regularly, you can detect errors quickly, and locate them more easily.
- Continuous integration is a necessity on complex projects and frequent releases.

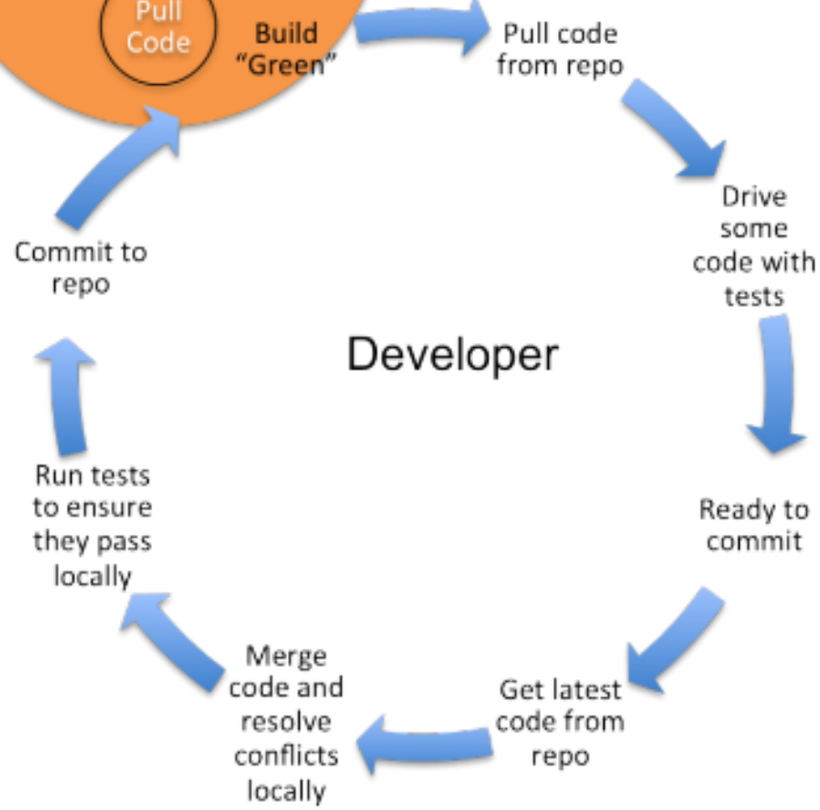


Continuous Integration Cycles

CI Server Build Cycle



CI Development Cycle



C.I. Tools

Version Control Systems

Git, SVN

Continuous Build Systems

Jenkins

Artifact Repositories

Nexus

Code quality inspectors

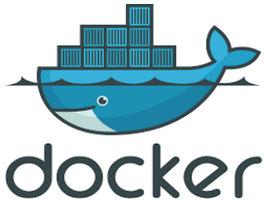
Sonar

Testing Frameworks

Junit, CppUnit, Robot Framework

Deployment Tools

Docker, Ansible



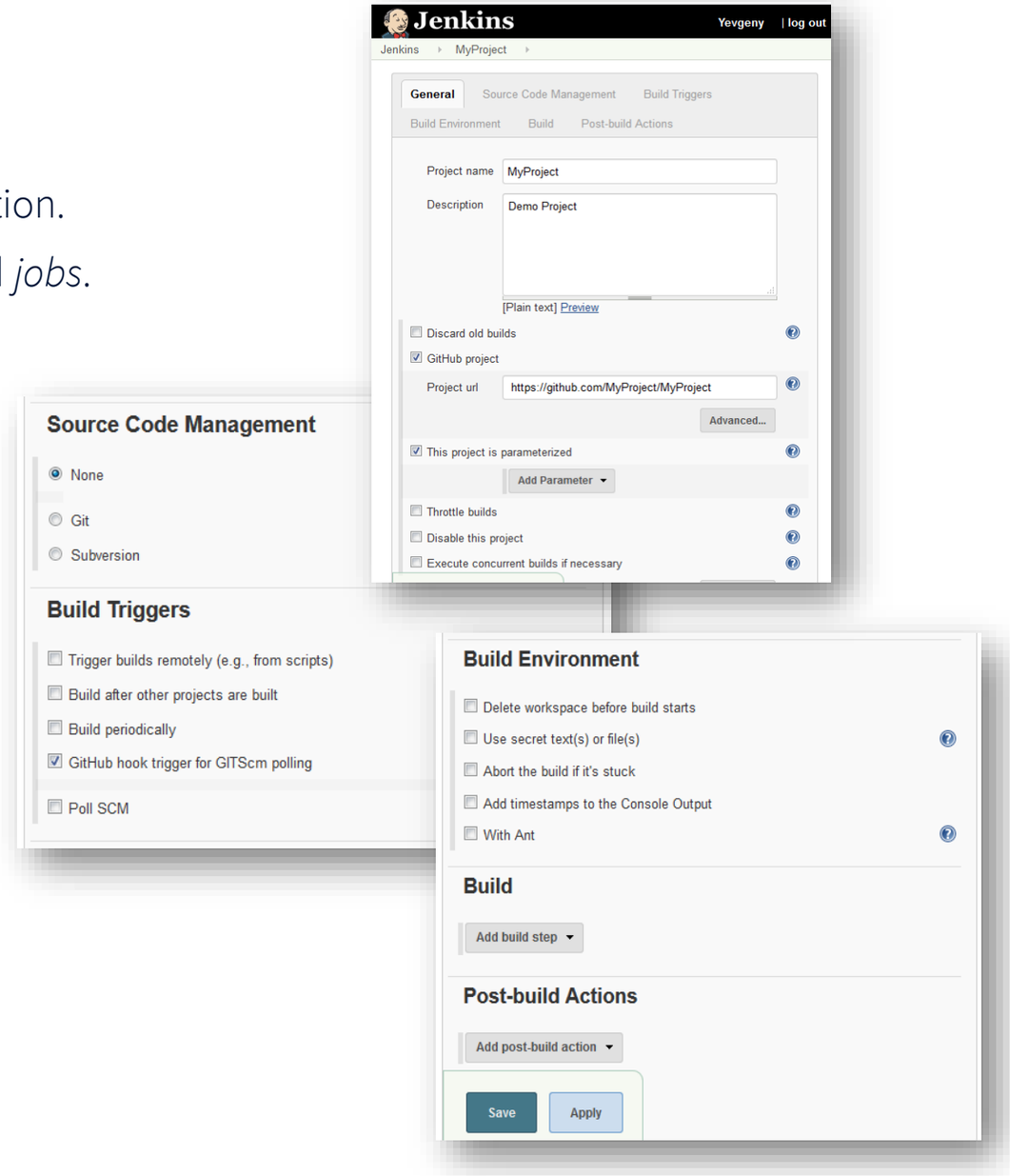
Jenkins Continuous Build System

Jenkins

Open source tool aimed to perform Continuous Integration and build automation.

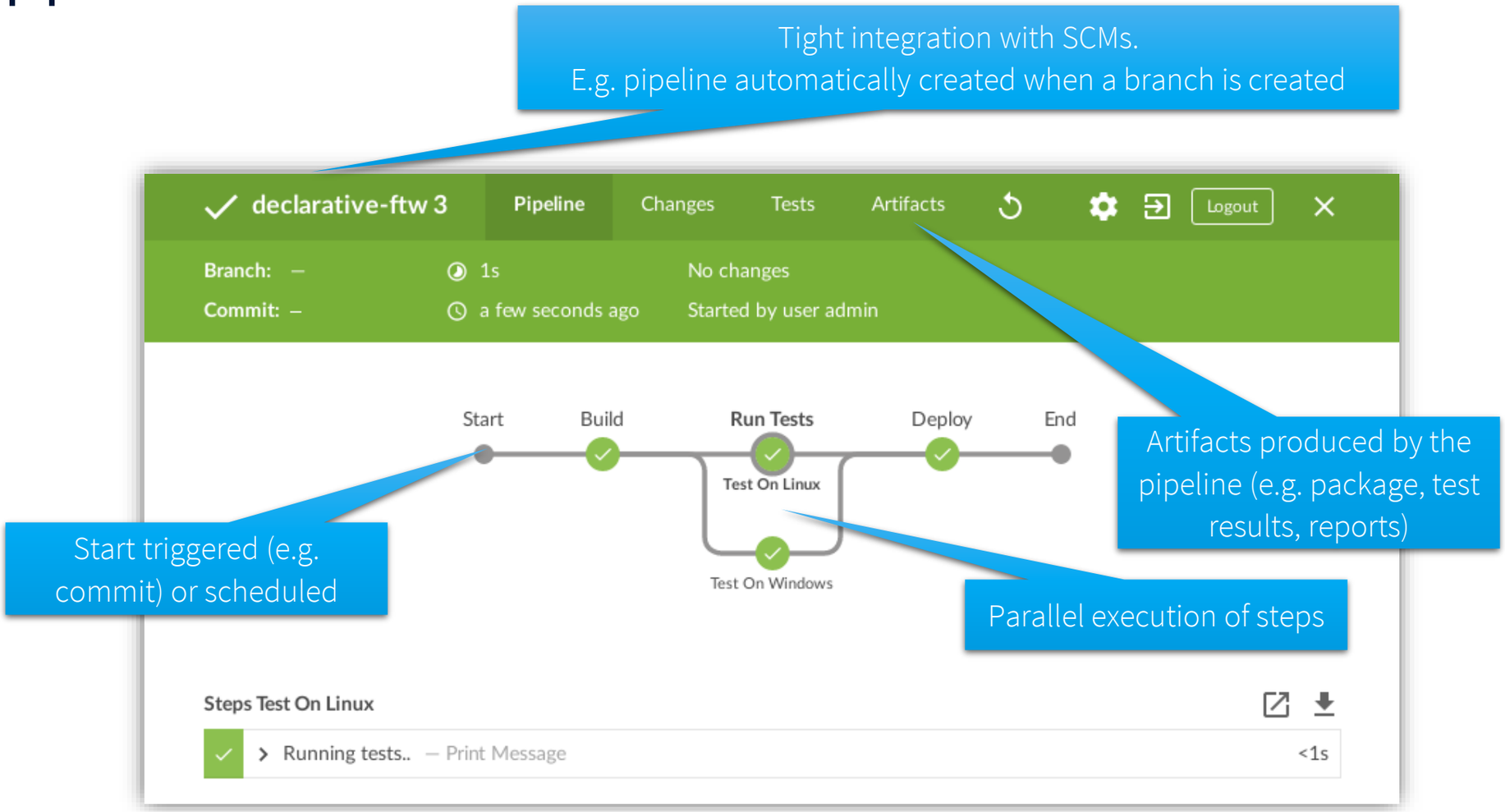
The basic functionality of Jenkins is to execute a predefined list of steps, called *jobs*.

- Jenkins job: sequence of tasks or steps in your build process, example:
  - Build the application
  - Verify the code quality
  - Run a shell script
  - Package the build result
  - Execute the integration and acceptance tests
  - Deploy / Release
- The jobs can be triggered in various ways, and chained together
  - e.g. of triggers: every day, after a commit, after another job execution



Jenkins

A Jenkins pipeline

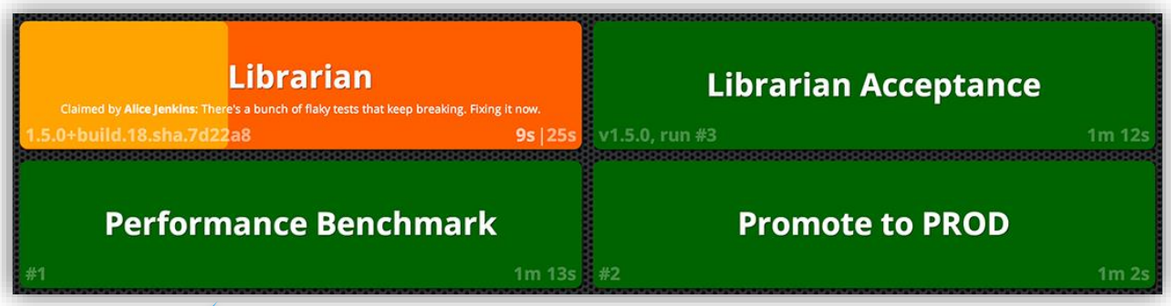


Jenkins

Jenkins Monitor

The Jenkins monitor displays in real-time the progress of the jobs being executed

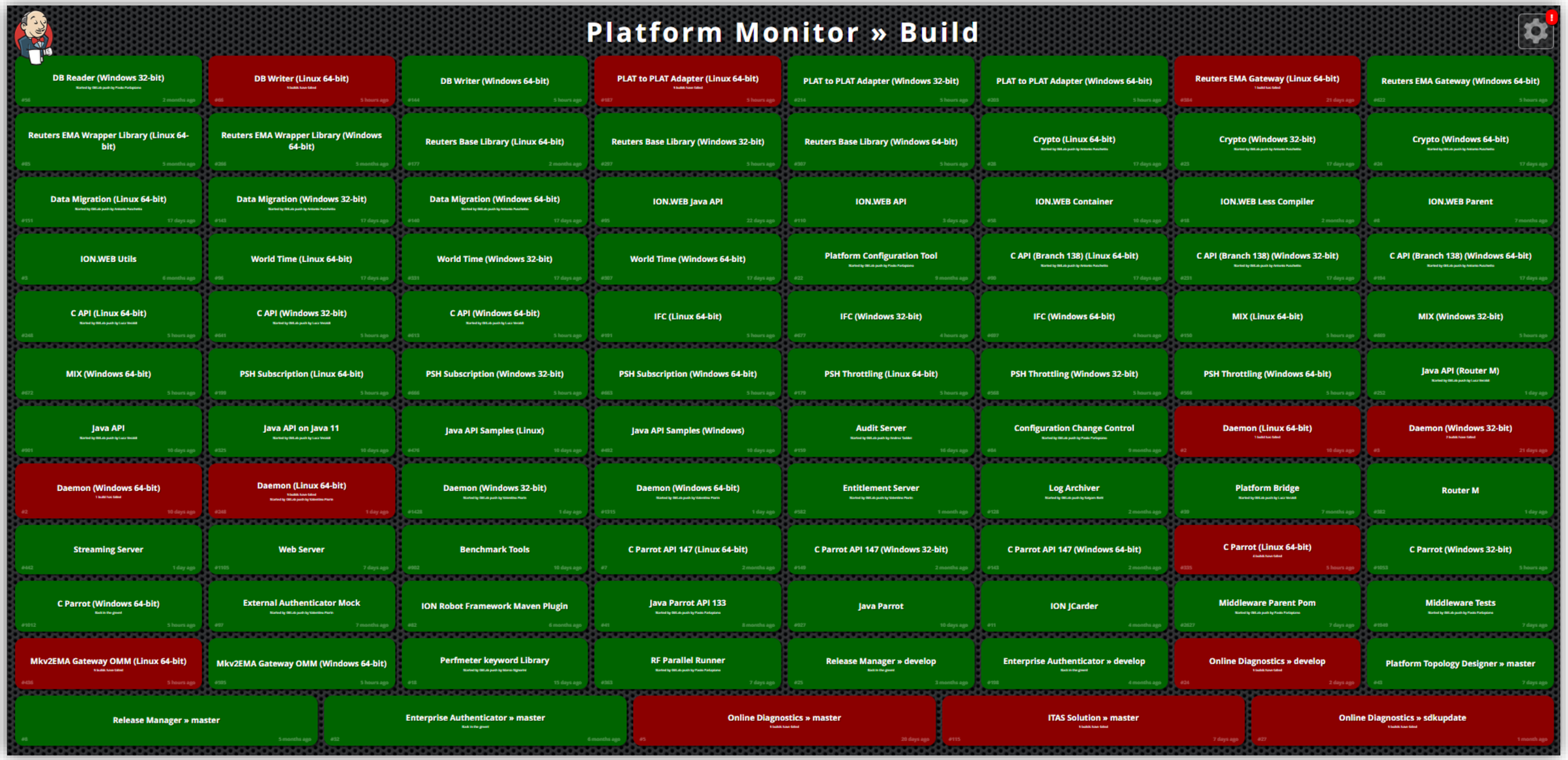
Usually, every development team has its own dedicated screens to monitor the status of the team's projects.



Red monitor??
No release!

Jenkins

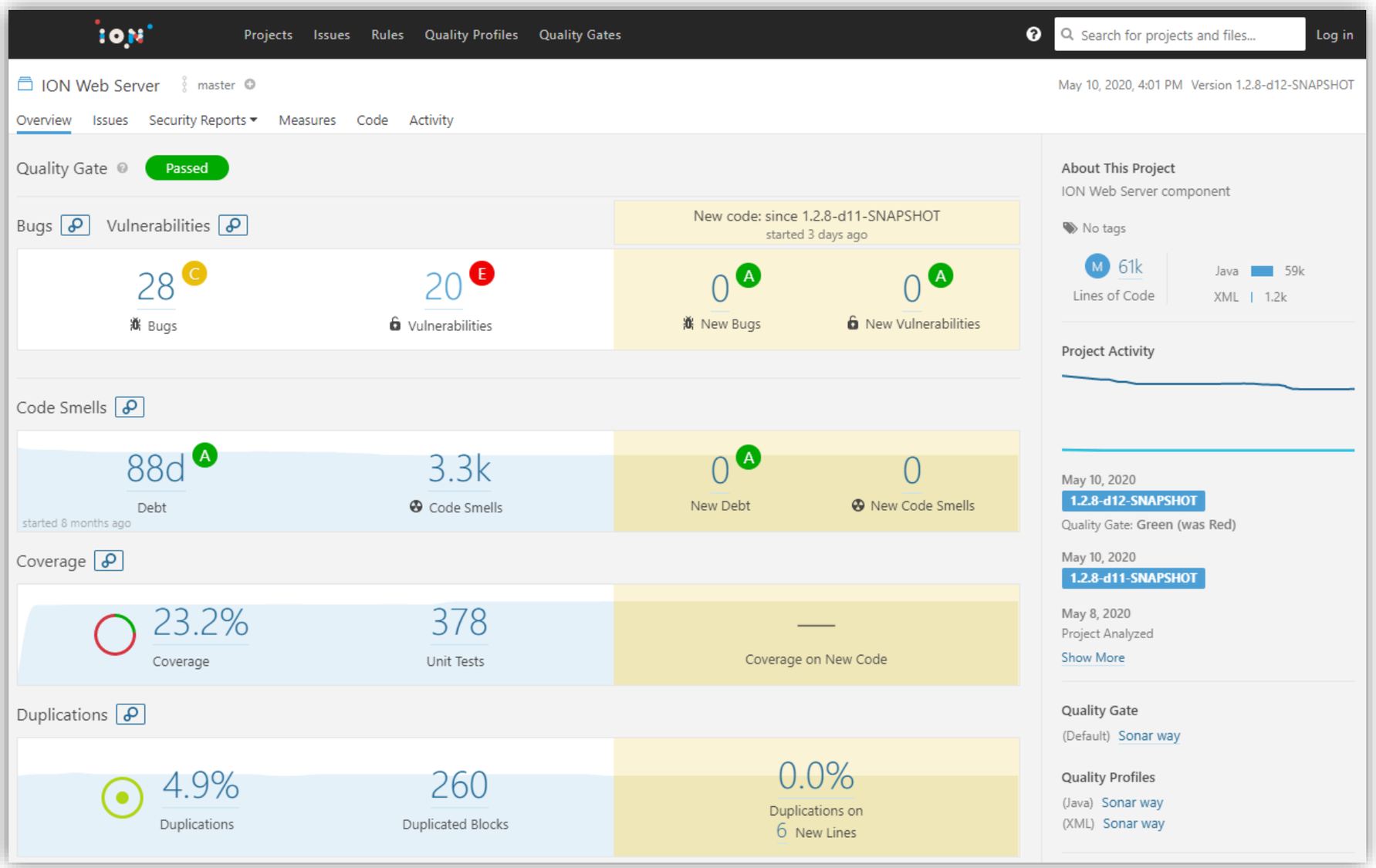
Continuous Integration and Testing Automation
Jenkins Monitor – ION Core Architecture Team – Builds



Continuous Integration and Testing Automation
Code quality checks – ION Web Server

Static Analysis

Dynamic Analysis



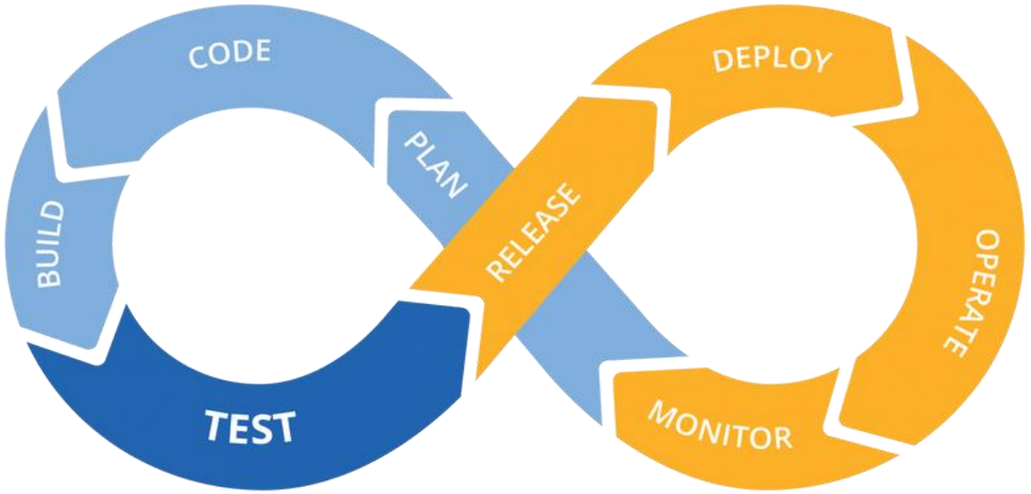
Testing Automation
&
Continuous Testing

What is testing automation?

It’s the use of software separate from the software being tested to control the execution of tests and the comparison of actual outcomes with predicted outcomes.

What is continuous testing?

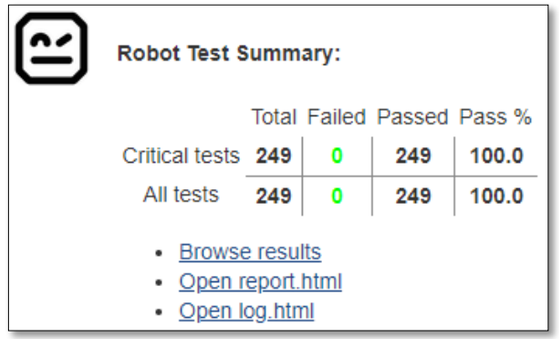
It’s the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate.



Continuous Testing with Jenkins and Robot Framework

Jenkins allows to execute Robot Framework tests

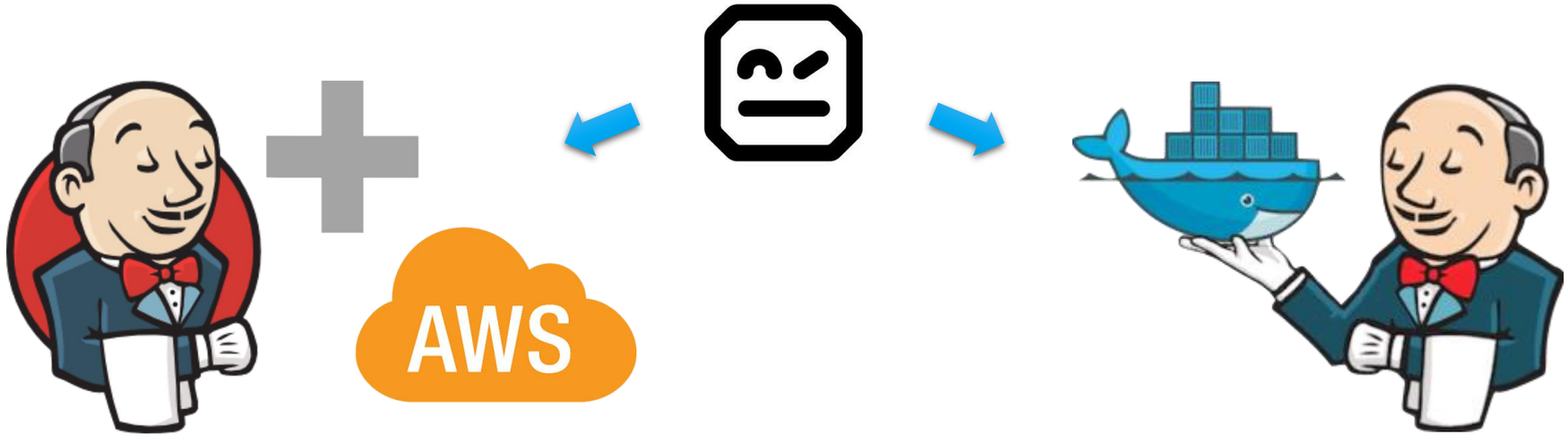
- The Jenkins job sets up the test environment and triggers the tests
- The outcome of tests is reflected in the job status
- Test reports are automatically attached and accessible on the Jenkins job
- Advanced testing environments can be easily setup
  - The slaves executing the tests can be dynamically allocated (e.g. cloud resources)
  - Containerization (e.g. Docker) is natively supported, allowing to spawn containers on demand for testing purposes



Robot Test Summary:

	Total	Failed	Passed	Pass %
Critical tests	249	0	249	100.0
All tests	249	0	249	100.0

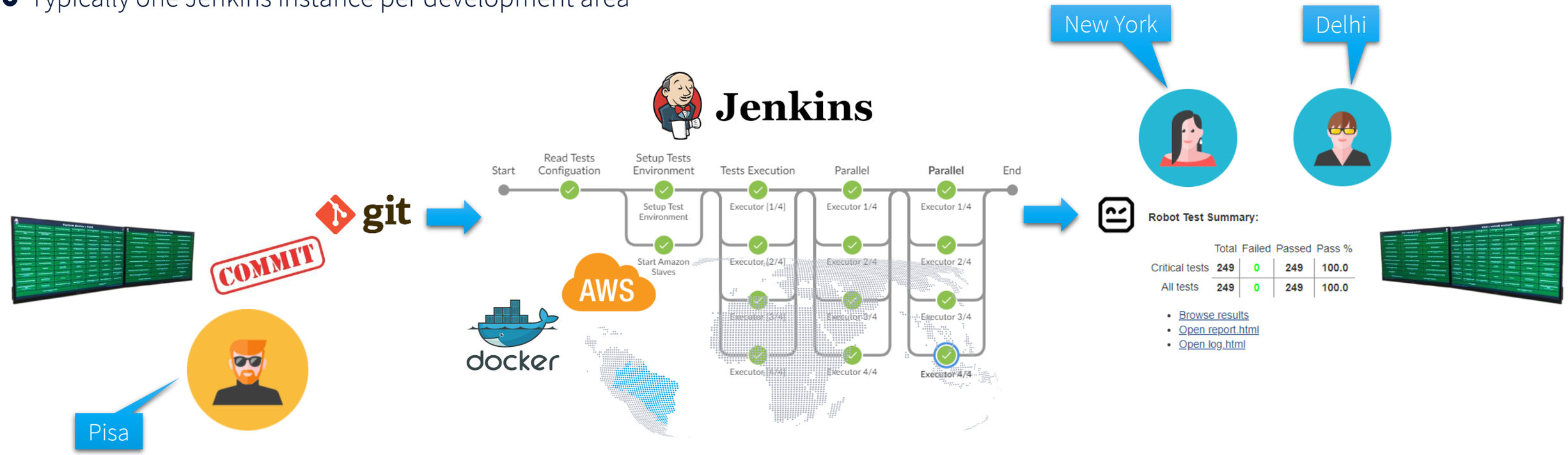
- Browse results
- Open report.html
- Open log.html



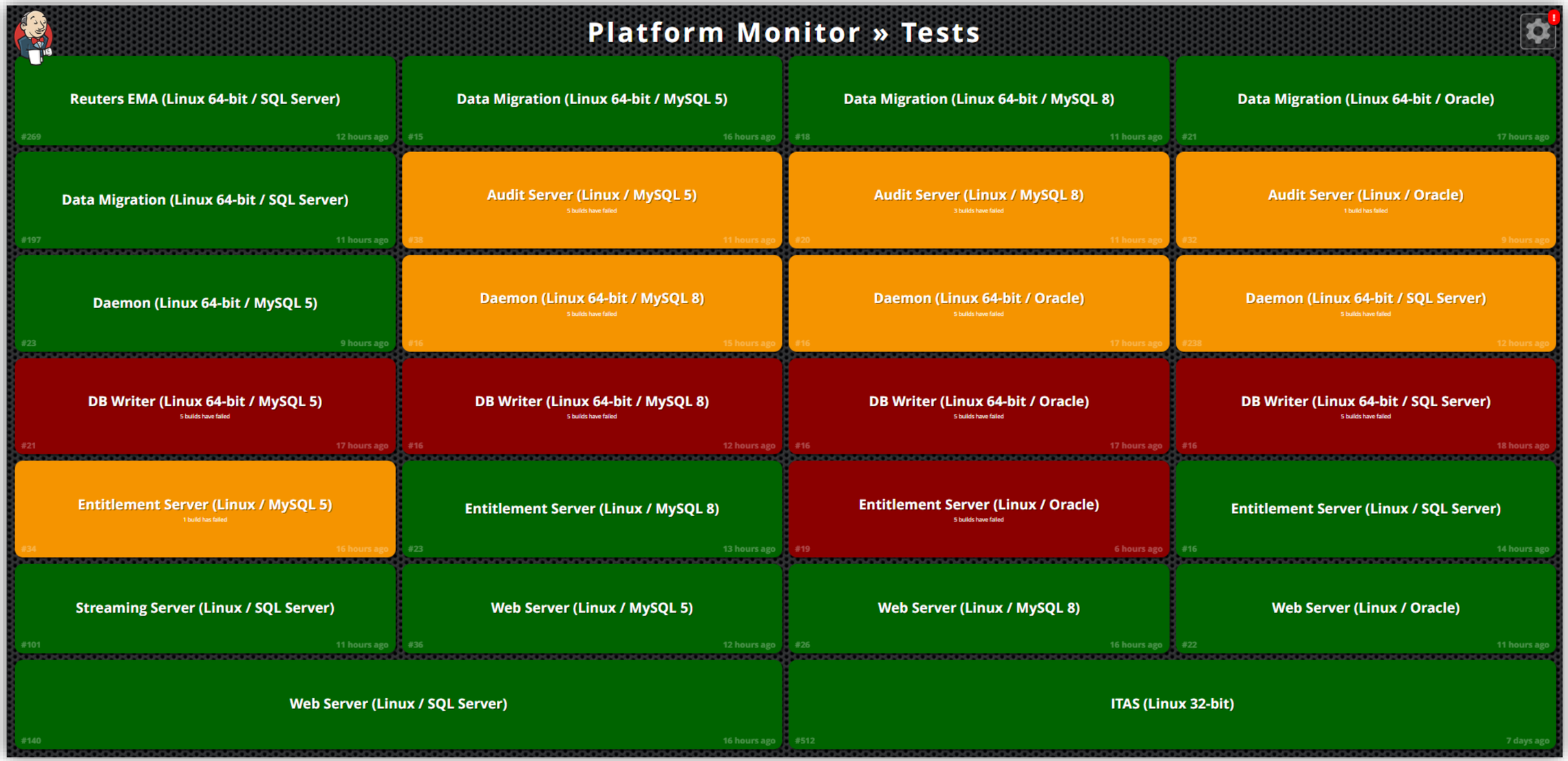
Testing Automation in ION

ION uses Jenkins and Robot Framework to automate the execution of tests

- Tens of thousands of tests executed automatically every day
- Several teams, geographically distant, share the same C.I. environment
- Heavy usage of the cloud and containerization to allocate the required hardware resources
- Typically one Jenkins instance per development area



Jenkins Monitor – ION Core Architecture Team – Acceptance Tests



Maintenance

Like the production code, the testing code requires maintenance

Real team scenario: several developers possibly changing the production code or adding new tests every day.
The probability of affecting negatively the C.I. is not negligible, some examples:

- On purpose backward incompatible changes in the production code
  - Luckily, they are very rare
- Regressions in the production code!
- Bugs in the tests
  - Instabilities
  - Not educated tests
- Performance issues
  - Not trivial tests often require a lot of time and resources to execute
  - Over the years, this can become a serious issue

Debugging required

What does maintenance mean

- Few supervisors check the status of the C.I. daily
  - For minor issues, they apply or delegate to other team members the required action points
  - For bigger issues, the team stops working on the production code (No tests?!? No developments are possible!)

Click here to apply