



*Didomi QA Challenge*  
*PART 2 – November 2020*

# TEST STRATEGY

# REQUIREMENTS

# CONSENT NOTICE PRODUCT PARTS

- SDKs (Web and Mobile) that are deployed on websites and mobile apps to show consent notices
- APIs that provide configuration to the SDKs and store events collected by the SDKs
- Administration portal ("Didomi Console") used to configure the SDKs through the APIs

# TEST STRATEGY SHOULD ANSWER TO:

- How you would design and execute the entire testing strategy for the consent notices product ?
- How you would manage a database of tests that grows with new features?
- What tests would be run manually vs automatically?
- How the work of creating and automating test cases would be distributed between product managers, engineers, etc.?
- How every project that is part of the product (backend, frontend, SDKs) would be tested and you would test the whole chain?

# TESTING STRATEGY FOR THE MOST COMMON SCENARIO

- Log into the administration portal
- Create or modify a consent notice
- Publish the changes
- Check that the changes are correctly propagated to your website or mobile app

# CONSENT NOTICE TEST STRATEGY

# TEST STRATEGY SUMMARY

1. Testing Scope
2. Test Approach and STLC
3. Test Environment
4. ~~Risk Analysis~~
5. ~~Review and Approvals~~

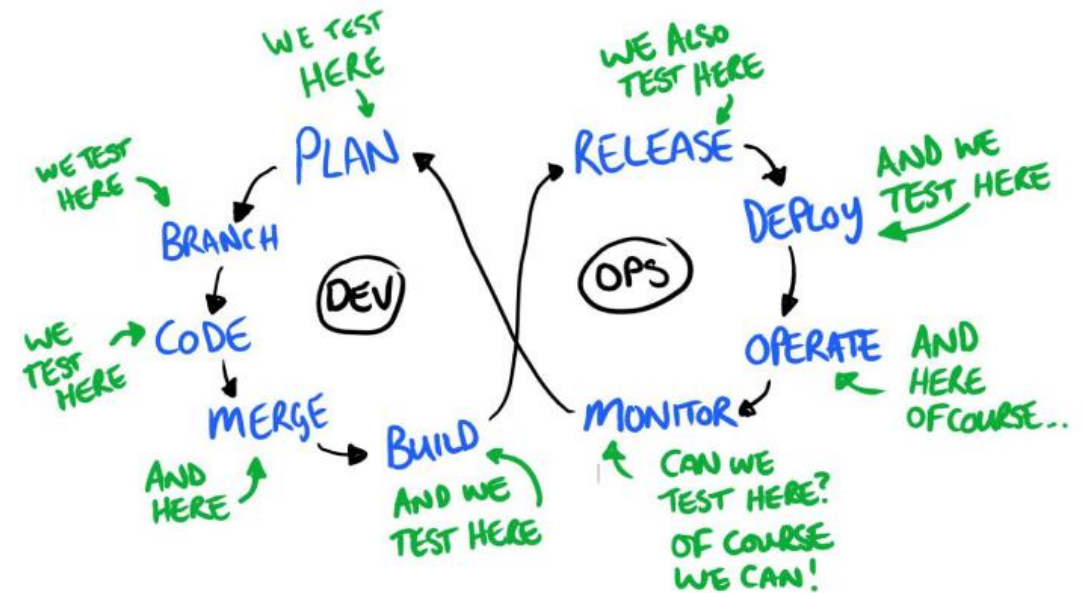
# TESTING SCOPE

- Didomi Console
  - Create/Modify/Delete a consent Notice
  - Configure SDKs
  - Get Events / board (?)
- SDKs:
  - Setup SDKs on **inhouse apps**
  - Get consent Notice configuration
  - Verify Consent Notice Behavior (showed, not showed, content, agree, decline ....)
  - Sent Events
- Backend : APIs and Databases
  - Integrity check
  - Performance
  - Mocked Data




# TESTING APPROACH AGILE /DEVOPS

- Have in mind
  - Agile testing, Definition of Done and Release Control (CI/Sonarqube)
  - Testing in a devOps world
- Lead a quality-driven transformation by
  - Sharing the test strategy and RTM
  - BDD / TDD
  - Measuring and metrics' analysis
- Small steps, big impact
  - Easy-to-implement steps that require minimal effort
  - Will have a significant impact on product delivery.

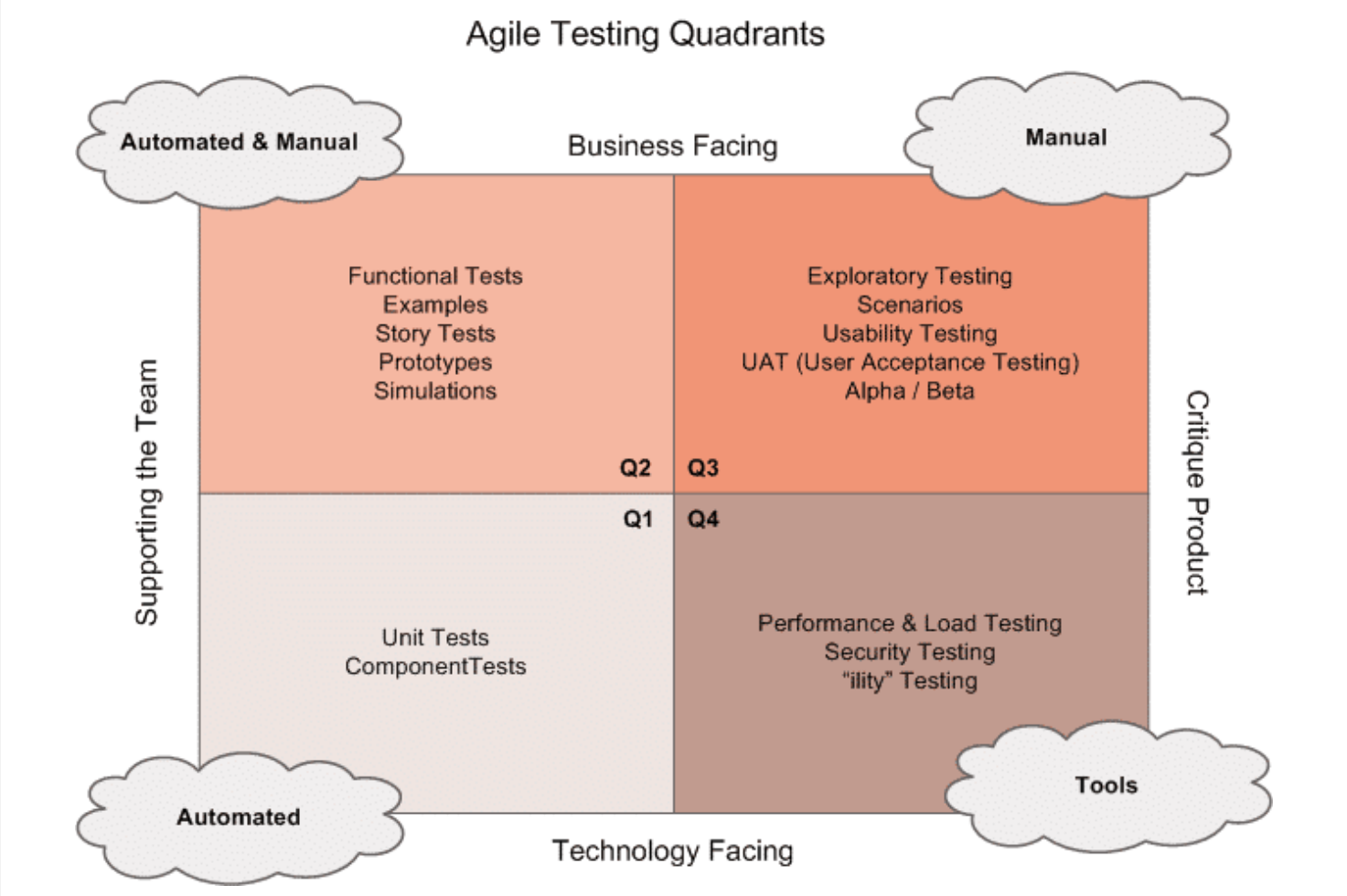


# REQUIREMENTS TRACEABILITY MATRIX (RTM)

PREPARED BY	Hajar	
DESIGNATION	Consent Notice Didomi Console	
DATE	24/11/2020	
TOTAL No. of BUSINESS REQUIREMENTS	5	
TOTAL No. of TEST CASES	10	

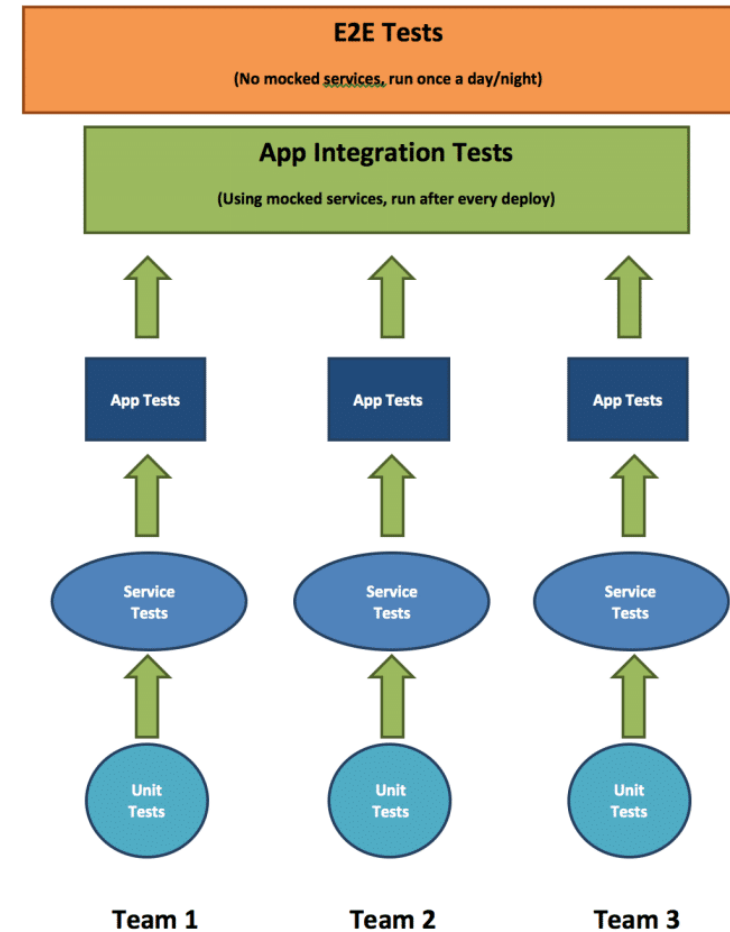
REQUIREMENT TRACEABILITY MATRIX		BUSSINESS REQUIREMENT ID's								
		BID001	BID002	BID003	BID004	BID005				
TEST CASE ID's	TID001	X								
	TID002	X								
	TID003	X								
	TID004		X							
	TID005		x							
	TID006			X						
	TID007				X					
	TID008				X	X				
	TID009					X				
	TID010					X				
	TID011					X				

# AGILE TESTING QUADRANTS



# APPROACH: AGILE TESTING LEVELS

- Test Automation Strategy for Multiple Agile Teams
  - Functional
    - Automated Unit Tests
    - Automated API or Service Tests
    - Application Testing
    - Integration Test
    - System / End-to-End Scenario Tests
    - Automated Acceptance Tests
  - Non functional
    - Performance Tests
    - Security Tests
    - Load tests (APIS/ Database / WebServer)
- Manual Testing
  - User Acceptance Tests or Business Acceptance Tests
  - Exploratory
  - Some UAT
- Regressions Tests



# APPROACH: TESTING LEVELS/ TEST TYPES

	WHY	WHO	WHAT	WHEN	WHERE	HOW
<b>Unit Testing</b>	To ensure code is developed correctly	Developers / Technical Architects	All new code + re-factoring of legacy code as well as Javascript unit Testing	As soon as new code is written	Local Dev + CI (part of the build)	Automated, Junit, Jasmine, mocha siesta
<b>API / Service Testing</b> <b>Integration</b>	To ensure communication between components are working	Developers / Technical Architects	New web services, components, controllers, etc	As soon as new API is developed and ready	Local Dev + CI (part of the build)	Automated, Soap UI, Rest Client JMeter
<b>Acceptance Testing</b>	To ensure customer's expectations are met	Developer / SDET / Manual QA	Verifying acceptance tests on the stories, verification of features	When the feature is ready and unit tested	CI / Test Env (dev branch, dev master)	Automated using BDD framework (Cucumber, Jbehave, Gauge)

# APPROACH: TESTING LEVELS/ TEST TYPES

(Type)	WHY	WHO	WHAT	WHEN	WHERE	HOW
<b>Performance / Security Testing</b>	To ensure servers's sanity	Developers/ SDET/ DevOps	Didomi console webservers API Database	As soon as new code is written	Staging Env	Healthchecks
<b>Load Testing</b>	To ensure servers can	Developers/ SDET/ DevOps	Didomi console webservers API Database	As soon as new API is developed and ready	Staging Env	Overload calling to Api/Db Overload connection to console
<b>System E2e</b> <b>UAT Regression</b>	To ensure the whole system works when integrated	SDET / Manual QA / Business Analyst / Product Owner	Scenario Testing, User flows and typical User Journeys	When Acceptance Testing is completed	Staging Env	Automated (Webdriver) Manual (UAT, Exploratory Testing...)
<b>3rd parties</b>	If it's down, a consent feature will fail	SDET	All external services that are component of Consent Notice Product.	Monitoring should run 7/7- 24/24	Production env	Have healthcheck (sensors, service map inhouse or external)

# APPROACH: REGRESSION PACKS

- **Smoke pack – Should be no more than 15 mins** only key high-level functionality to make sure the application is stable enough for further development or testing (load, access, few key scenario).
- **Functional Regression Packs**, which is meant to check the functionality of the application in more detail than the smoke test.
- **End-to-End Regression Pack**, which tests the whole application as a whole. The aim of these tests is to ensure that various parts of the application which connect to various databases and third-party applications work properly.
- **Full regression pack – should be no more than 1 hour**, only high-level functionality to make sure the application is stable enough for further development or testing

# TEST ENVIRONMENTS

- Manual Tests VS Automated Tests
- Test Database management
- Product parts' tests exécution
- Test Automation Roles Distribution
- Testing metrics
- ~~Testing tools~~





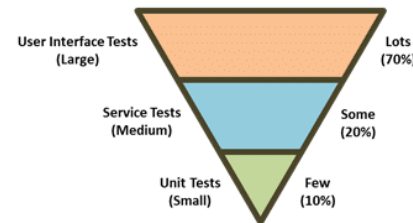
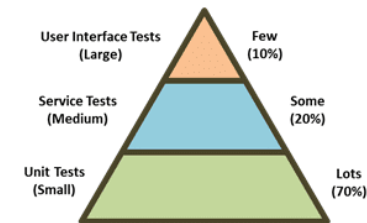
**WHAT TESTS WOULD  
BE RUN MANUALLY  
VS AUTOMATICALLY?**



# MANUAL VS AUTOMATED - TESTING VS CHECKING

- Avoid comparison between manual and automated testing.
- Testing:
  - During manual testing, tester's brain is engaged
  - Tester can alter the flows during the manual testing (exploratory testing, monkey testing...)
- Checking:
  - Automated testing relies on Tests Data
  - From an input, we expect a specific result

**An upright pyramid is often a good complement to manual testing.** →



← **Be careful of upside-down pyramids.**  
**They often fail to meet expectations.**

MANUAL	AUTOMATED
Tests that you will only run only once. The only exception to this rule is that if you want to execute a test with a very large set of data, even if it's only once, then it makes sense to automate it.	Unit Testing
User experience tests for usability (tests that require a user to respond as to how easy the app is to use).	API / Service Testing
Tests that need to be run ASAP. Usually, a new feature which is developed requires a quick feedback so testing it manually at first	Integration
Tests that require ad hoc/random testing based on domain knowledge/expertise - Exploratory Testing	Acceptance Testing
Intermittent tests. Tests without predictable results cause more noise than value. To get the best value out of automation the tests must produce predictable and reliable results in order to produce pass and fail conditions.	Performance / Security Testing
Tests that require visual confirmation, however, we can capture page images during automated testing and then have a manual check of the images.	Load Testing
Test that cannot be 100% automated should not be automated at all, unless doing so will save a considerable amount of time. For instance SDK setup on inhouse App.	System/E2E UAT Regression





**HOW YOU WOULD  
MANAGE A  
DATABASE OF TESTS  
THAT GROWS WITH  
NEW FEATURES?**



# AUTOMATE TEST DATA

- Test data is mandatory to be able to test but :
  - Time waiting for test data refreshes is a significant aspect of software development time (including testing)
  - Refresh Data request can be unnecessarily complicated and thus time consuming process
- Why does it take so much time?
- What to do ?
  - Refresh data set via a self-service portal.
  - integrate with tools to automate the test data provisioning
    - and subsetting and masking data can be automated as well (PII handling may need it).

# CHALLENGE : SIMULATES REAL-WORLD CONDITIONS

- Challenge : build a test database that faithfully represents production systems that enable to cover as many real-life test cases as possible.
- It's difficult:
  - Huge diversity of interactions between users and live production systems
  - Wide range of test cases that QA needs to cover.
- Test cases include differences in:
  - End cases
  - Data combinations
  - Errors in data
  - Methods of entering data
  - Data boundaries (for example, what happens if you try to push 3,000 characters into a 30 character field?)
  - Corrupt data scenarios (data packets do sometimes get lost or corrupted in transition, and computers can crash suddenly)

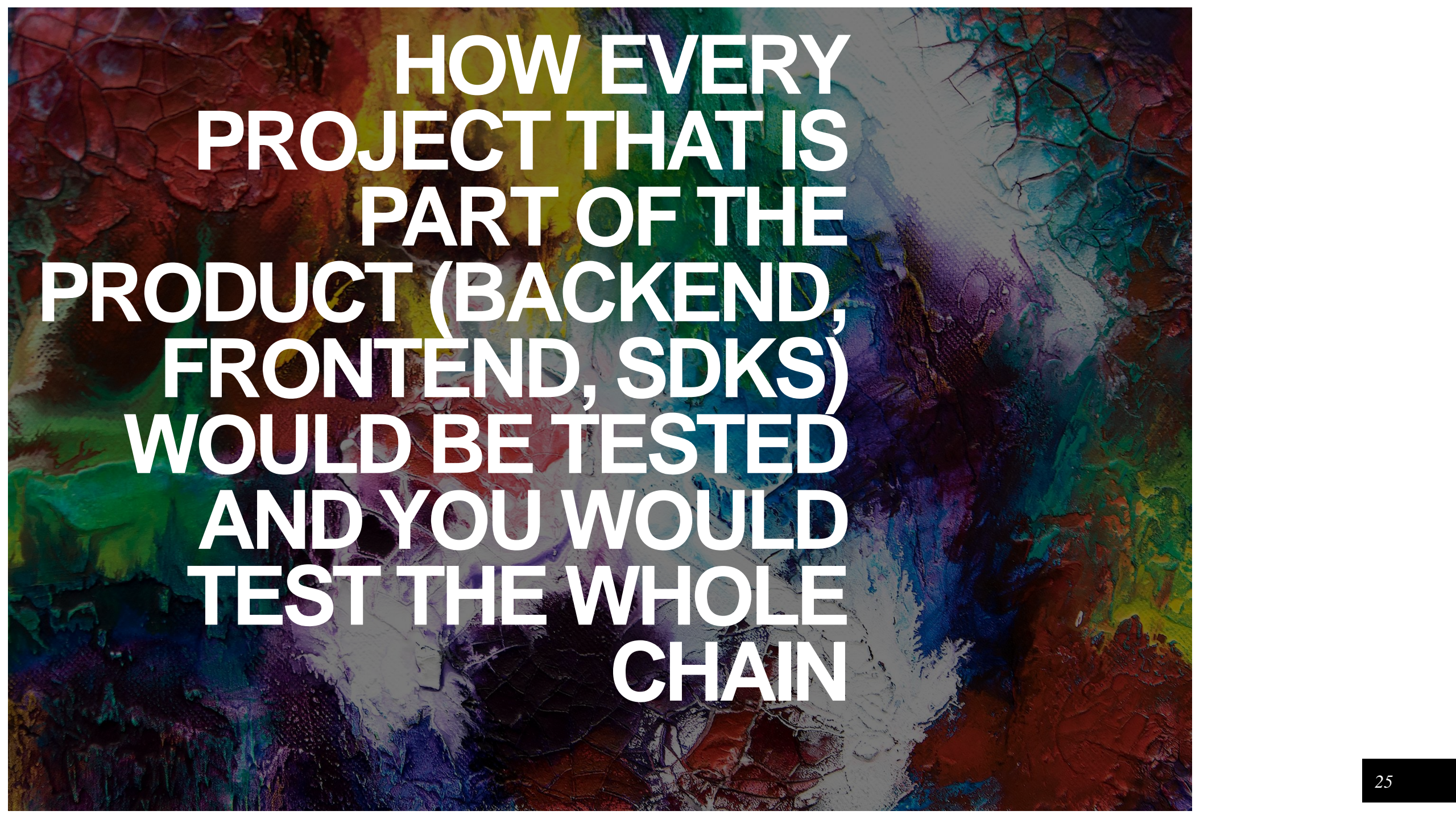
# BUILDING TESTS DATABASE

- Global Effort Support, Product, Dev, DB team and QA
- Test Data Creation can be done:
  - manually
  - using a data generation tool
  - can be retrieved from existing production environment.
- Tests Database Key principles:
  - Synthesize data
  - Use a customer's production database => go through this process whenever database schemas are updated
  - Simulate erratic behavior to generate unexpected end cases
  - Use performance engineering to simulate heavy loads
    - Fill with a large amount of data
    - Numerous concurrent users

# TEST DATA MANAGEMENT

- Data needs to be managed properly in order to be useful for tests. Test data management can be divided into 4 steps:
  1. Data Knowledge
  2. Subset Datasets
  3. Masking Data
  4. Automate Test Data
- Data mapping is the key to data management [ETL Mapping Sheet.xlsx](#)



The background is an abstract, textured surface with a mix of dark and vibrant colors including deep reds, blues, greens, and purples. A semi-transparent dark grey or black rectangular overlay covers the majority of the image, serving as a backdrop for the white text.

**HOW EVERY  
PROJECT THAT IS  
PART OF THE  
PRODUCT (BACKEND,  
FRONTEND, SDKS)  
WOULD BE TESTED  
AND YOU WOULD  
TEST THE WHOLE  
CHAIN**



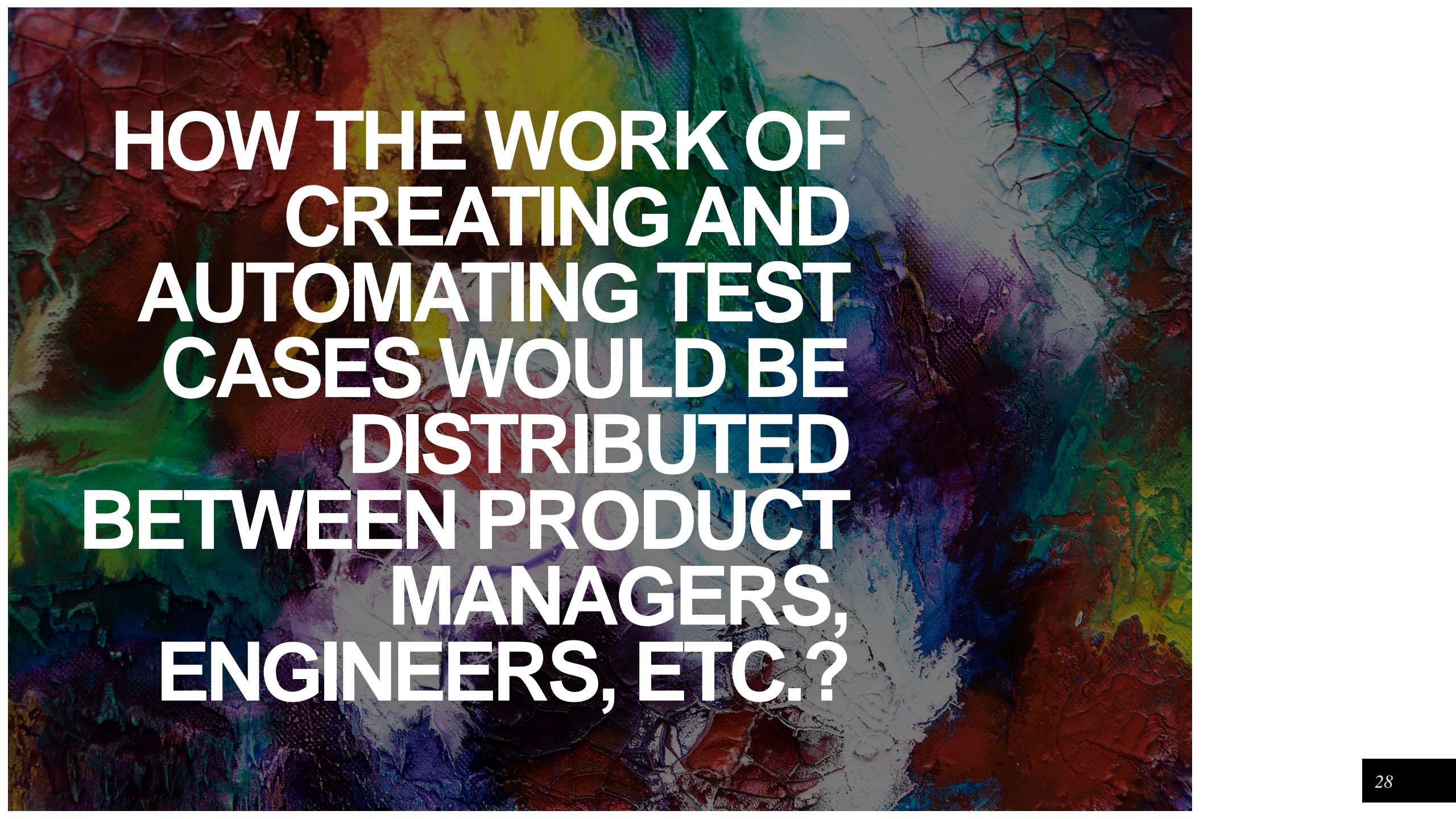
# TEST EXECUTION BY PART

- For each part, Tests are included in the development cycle and triggered on CI
- Tests success is mandatory for release engineering  
Dev Check => Preprod Check => Staging Check => Prod Check
- **For all product parts: unit tests, integration tests, system**
- **Frontend: Didomi console**
  - UI Testing,
  - Perf/load testing
  - AT, UAT
- **Backend: APIs & Database**
  - APIs Testing
  - Database Testing
  - Perf/load testing
- **SDKs:**
  - Manual tests with Sample App : Setup (once), Consent Notice behavior

# WHOLE CHAIN

- Automate scope's scenarios :
  - Integration testing
  - System testing
  - E2E test
  - UAT
- Create Regressions packs



The background of the slide is a vibrant, abstract composition of various colors including reds, oranges, yellows, greens, blues, and purples. These colors are layered and blended in a way that creates a sense of depth and movement. Overlaid on this colorful background is a dark, semi-transparent layer that serves as a canvas for the white text. The text is arranged in a centered, stacked format, with each word or group of words on a new line. The font is a clean, modern sans-serif typeface, which makes the text highly legible against the complex background.

**HOW THE WORK OF  
CREATING AND  
AUTOMATING TEST  
CASES WOULD BE  
DISTRIBUTED  
BETWEEN PRODUCT  
MANAGERS,  
ENGINEERS, ETC.?**



# APPROACH: ROLES AND RESPONSABILITY

	Dev Backend	Dev Frontend	Data Engineer	QA/ AUTOMATION TEAM	PO/PM	OTHER
Unit Tests	X	X	X			
Integration / API or Service Tests	X	X	X			
Application Testing	X	X	X			
Integration Test	X	X	X	X		
End-to-End Scenario Tests Automated UAT				X	X	
<u>Manual Testing :</u> <ul style="list-style-type: none"><li>• Exploratory</li><li>• User Acceptance Tests</li><li>• Business Acceptance Tests</li></ul>				X	X	X
Regressions Packs	X (run)	X (run)	X (run)	X (create, run and analyse)	X (run)	X (run)

***“Quality” ceased to become the sole responsibility of the testers and became a shared responsibility of everyone involved in developing and delivering the product.***

# TEST METRICS : QA

Process	Product
Test Case Preparation Productivity	Error Discovery Rate
Test Design Coverage	Defect Fix Rate
Test Execution Productivity	Defect Density
Test Execution Coverage	Defect Leakage
Test Cases Passed	Defect Removal Efficiency
Test Cases Failed	
Test Cases Blocked	

# « AGILE » TEST METRICS

- Direct user feedback, capturing satisfaction and engagement of real users (hard to collect, but high value)
- The velocity of story delivery (easy to collect, medium value)
- Number of customer-reported defects (hard to collect, high value)
- Costs of delivery (hard to collect, high value)
- Team satisfaction (medium difficulty, high value)

**USER  
PERSPECTIVE:  
MOST COMMON  
SCENARIO**

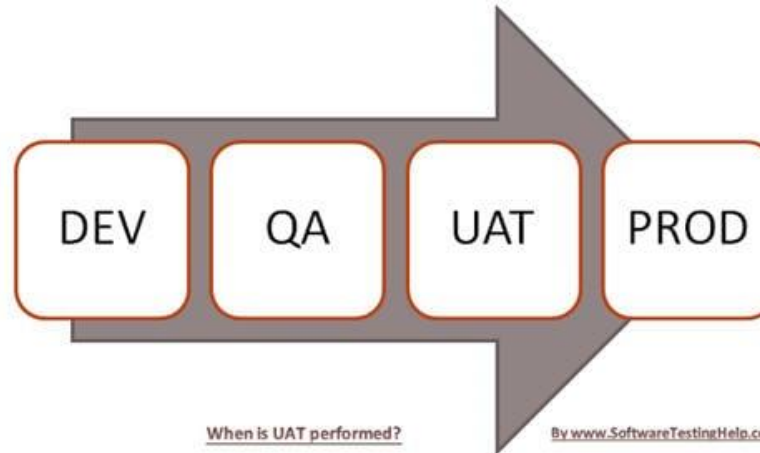


# TEST STRATEGY

- Should rely on Positive low level testing (unit Testing, integration, system, api, db)
- Audit real customers one for every type of SDKs :
  - Environment used,
  - People using the product
- Get help from support team to get Consent Notice Product FAQ
  - Understand how they use it
  - What issues they encounter
  - How they're able to report it
- Define User Acceptance Test
  - Rates tests priority => regression tests
- Run UAT, results follow up and UAT updates

# DEFINE & RUN UAT

- What are the key features to be tested ? What are dependancies between projects?
- Writes test cases for each features including :
  - Common behavior
  - Erratic behavior
  - Data boundaries cases
- DoD for each feature
- Define Quality Gates
  - Mandatory features
  - Workaround
  - Validation rules



# UAT DEFECT MANAGEMENT

- UAT Defect should trigger impediment on sprint to be fixed
- It should be maintained as long as the product evolves
  - Frequent customer survey to follow up their needs, issues and questions
  - Improve UAT according to surveys and support team
  - Include new types of customer (new type of SDK, new region)

# THANKS FOR LISTENING 😊

Hajar ZAROUAL

📞 +33 621 93 02 36

✉ *Zaroual.h@protonmail.com*

🔗 *[MissHajar \(MissH\) \(github.com\)](#)*