ACCEPTANCE-TEST AUTOMATION WITH KARATE

BJÖRN BESKOW

CADEC 2024-01-18 & 2024.01.24 | CALLISTAENTERPRISE.SE

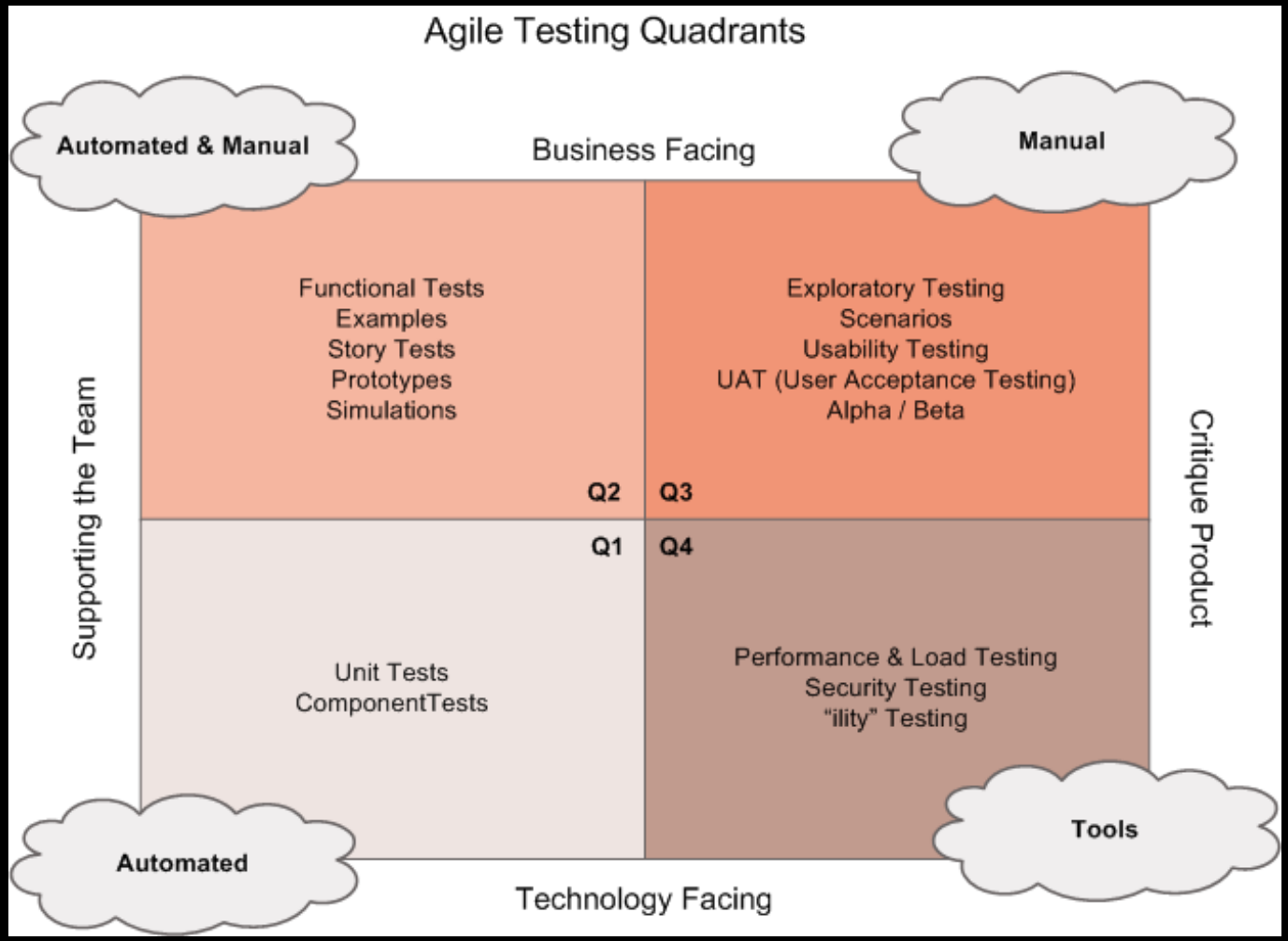
CALLISTA

AGENDA

- Background
  - Acceptance Test Driven Development
  - Previous Tools
- Karate Framework
  - API Testing
  - Web UI Testing
  - Event driven Testing
  - Hybrid Testing
- Conclusions

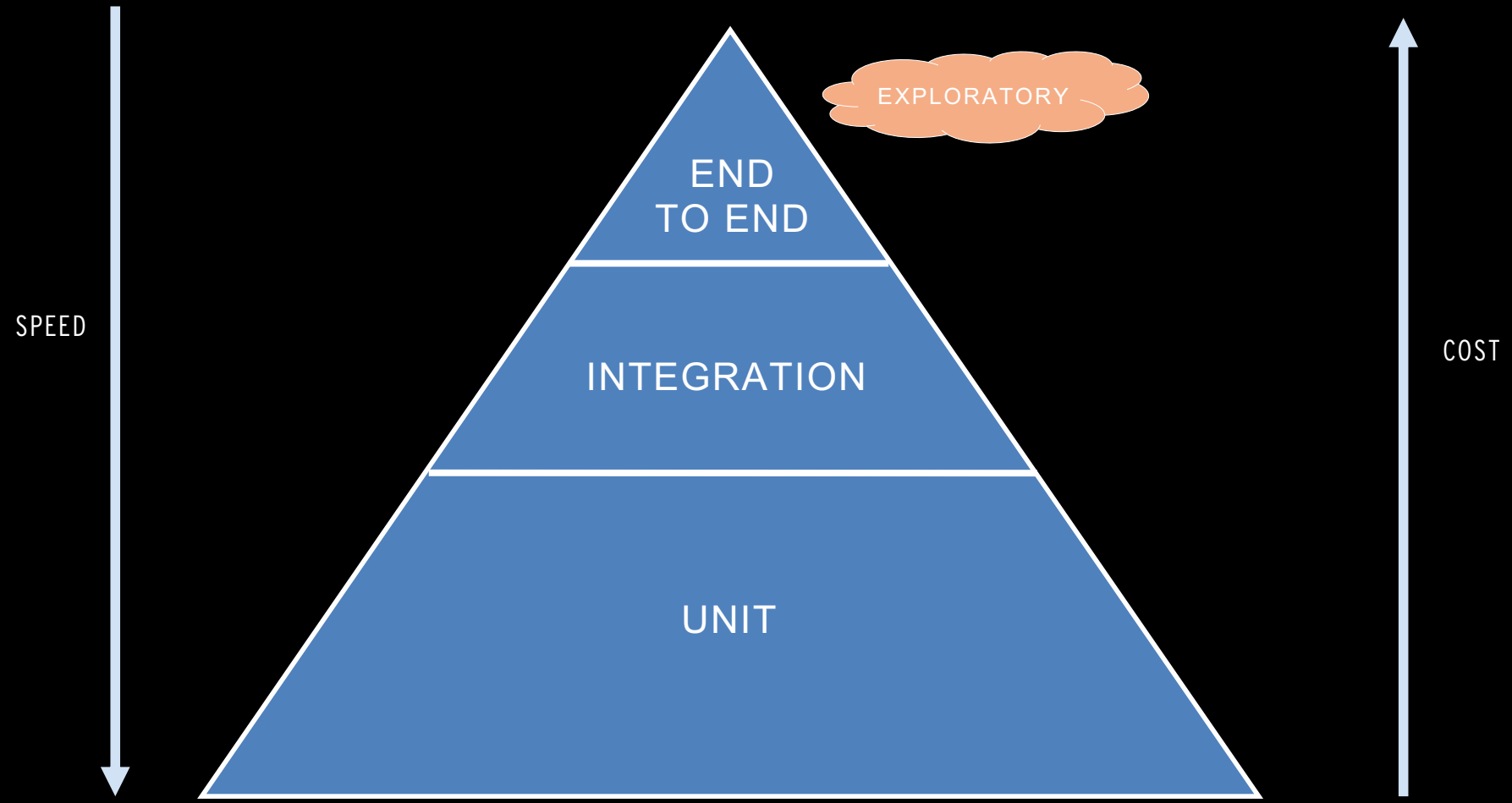
"Agenda by Abel, & MQ MSK WCA LosAngeles Graffiti Art" by anarchosyn

THE PURPOSE OF TEST AUTOMATION



SOURCE: LISA CRISPIN, 2011.

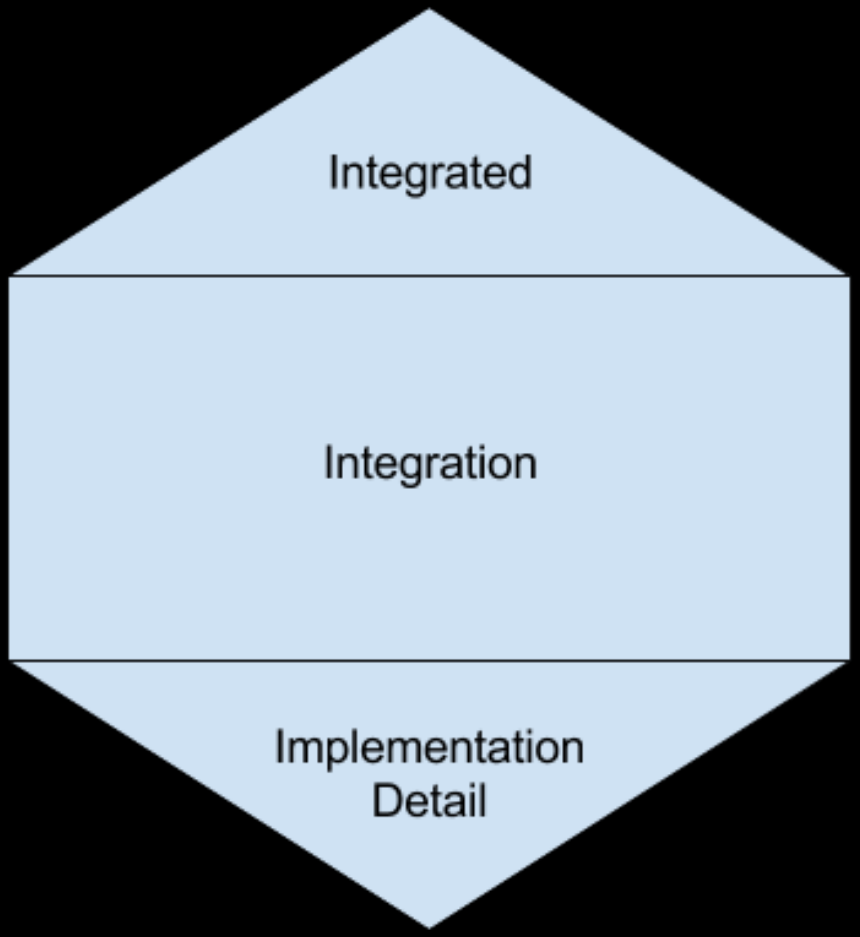
TRADITIONAL TESTING PYRAMID



SOURCE: JANET GREGORY, 2009.

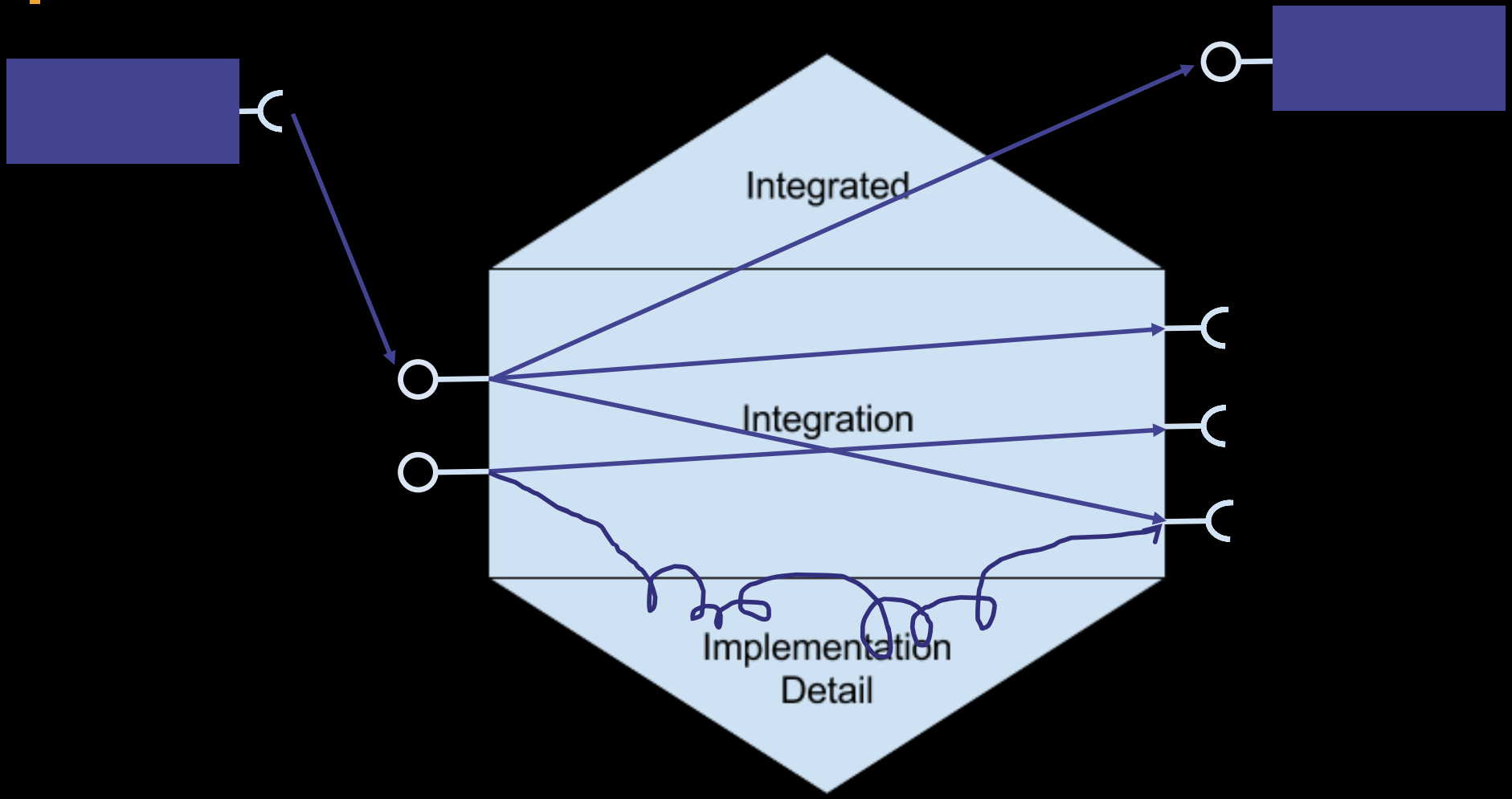
CALLISTA

HONEYCOMB TESTING STRATEGY



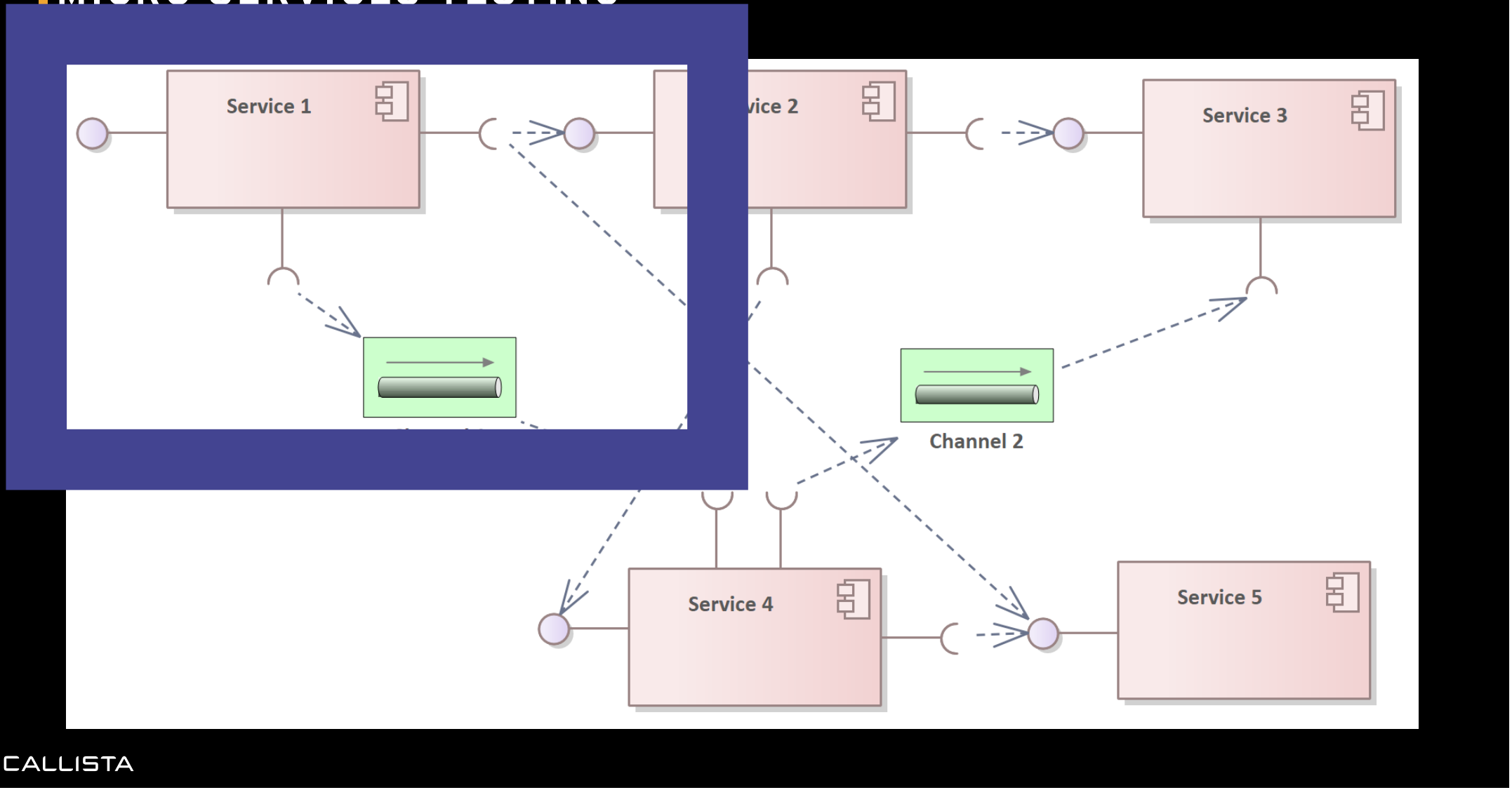
HTTPS://ENGINEERING.ATSPOTIFY.COM/2018/01/TESTING-OF-MICROSERVICES/

HONEYCOMB TESTING STRATEGY

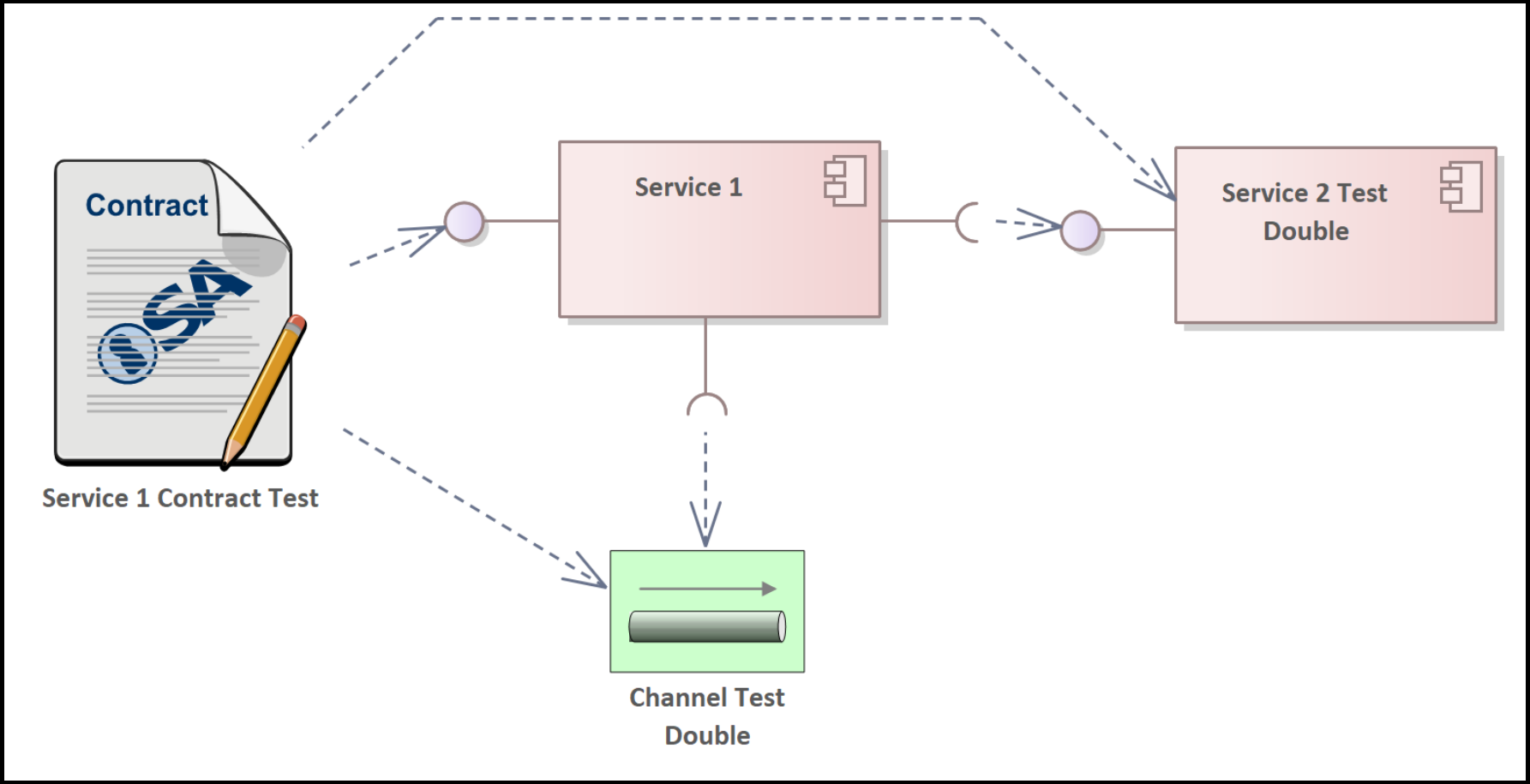


HTTPS://ENGINEERING.ATSPOTIFY.COM/2018/01/TESTING-OF-MICROSERVICES/

MICRO SERVICES TESTING



CONTRACT TESTING



BEHAVIOUR DRIVEN DEVELOPMENT

Business people wants to express the required behaviour, not necessarily write tests

BEHAVIOUR DRIVEN DEVELOPMENT

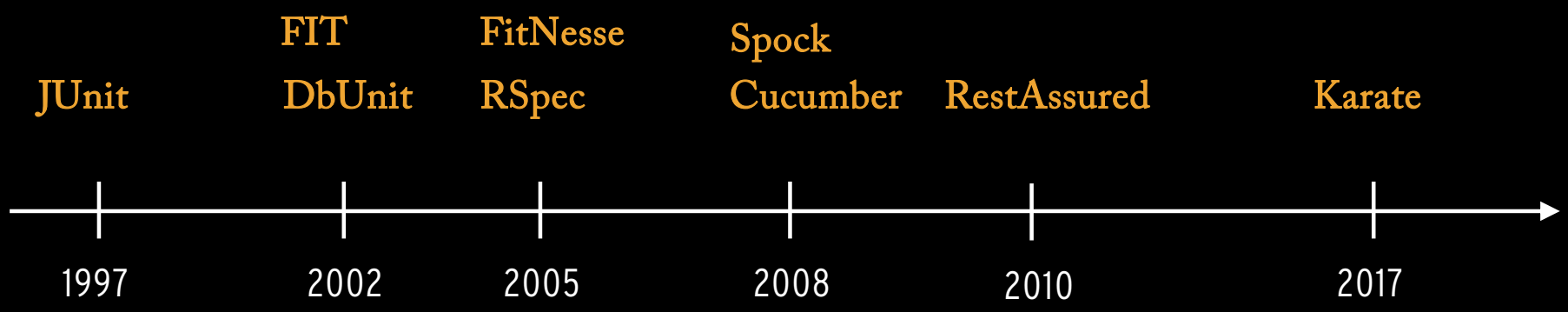
State required behaviour in a precise enough way, using the vocabulary of the business

State required behaviour using examples

CHALLENGES

- Readability
- Expressiveness
  - Ability to abstract/encapsulate/"hide" low level details
- Repeatability
- Robustness

LEGACY


JUnit
FIT
DbUnit
FitNesse
RSpec
Spock
Cucumber
RestAssured
Karate
1997
2002
2005
2008
2010
2017

KARATE

- Originally developed by Peter Thomas at Intuit
- Released February 2017 as Open Source
- Version 1.0.0 in March 2021, now at version 1.4.1
- Originated as a DSL for writing API acceptance tests, it has grown into a more full-fledged test automation framework
- Implemented in Java, runs on GraalVM

KARATE MISSION

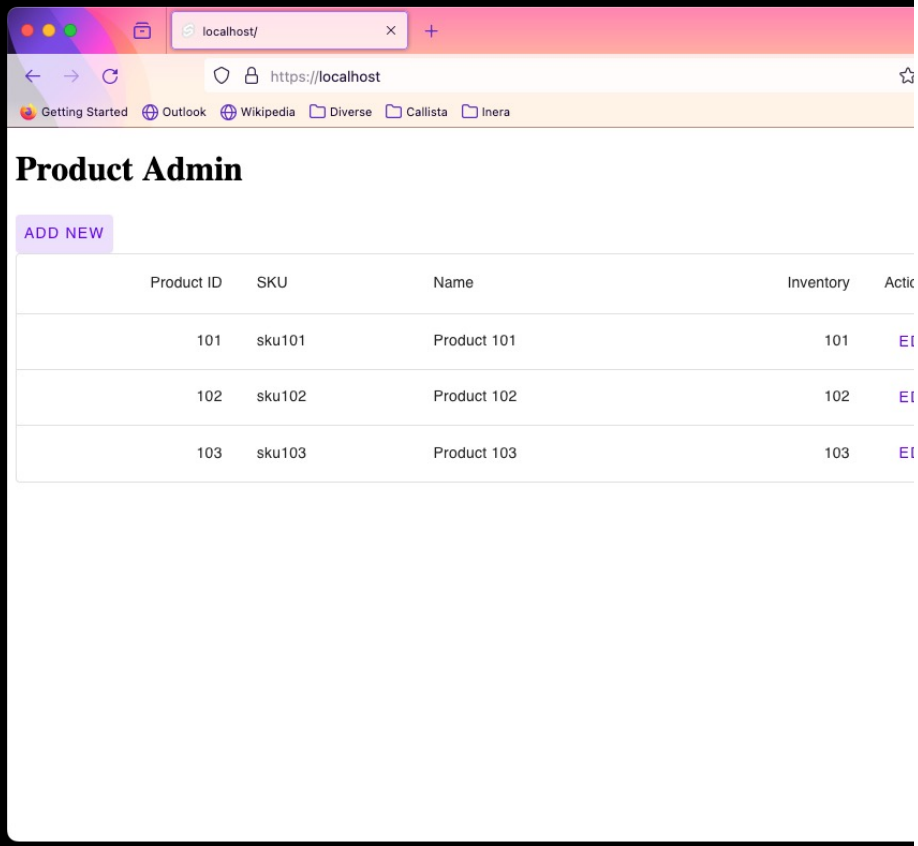
”Karate strives to reduce the entry barrier to writing a test and more importantly - reduces the friction to maintain a test, because of how readable tests become.”

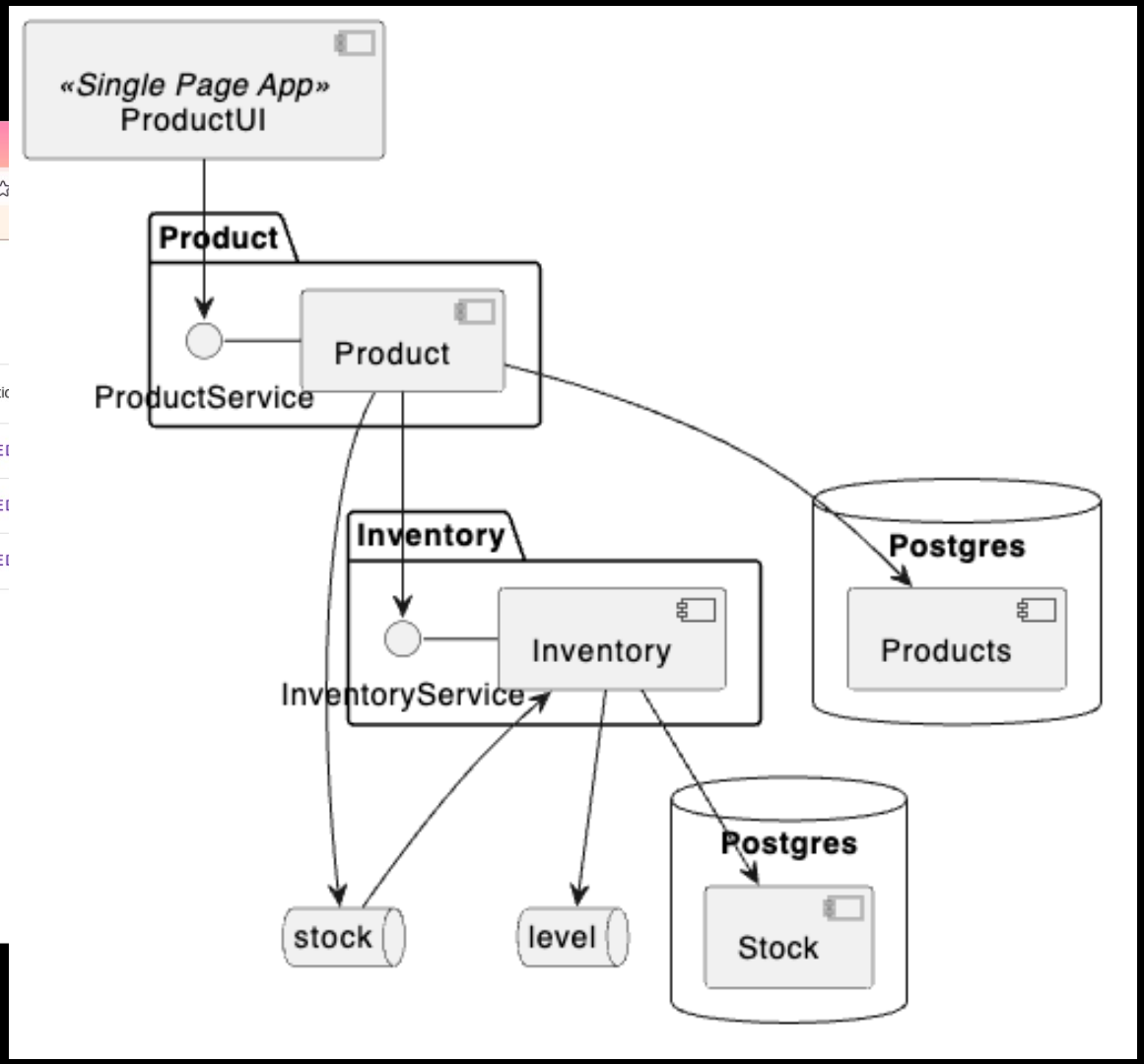
Peter Thomas

KARATE BY EXAMPLE: DEMO!

- API testing
  - Power Assertions
- Web UI Testing
- Data-driven tests
- Extending the DSL
  - Javascript functions
  - Java Interop
- Testing Event Driven APIs
- Managing Test Data
- Hybrid User Journey Testing

KARATE BY EXAMPLE: DEMO!





FUZZY MATCHER KEYWORDS

- Expected datatypes — `#string`, `#number`, `#boolean`, `#array`, `#object`
- Presence of a value — `#present`, `#notpresent`
- Null or not null — `#null`, `#notnull`
- UUID format — `#uuid`
- Regex matching— `#regex <string>`
- Advanced array validation — `#[Num] EXPR`
- Ignore or skip comparison — `#ignore`
- Optional fields with extra # — `##string`, `##number` etc.

DOMAIN SPECIFIC LANGUAGE (DSL)

```
Scenario: create a product
  Given url product_url
  And path '/products/'
  And def articleId = 'new'
  And def name = 'a new product'
  And request {name: #(name), articleId: #(articleId)}
  When method post
  Then status 201
  And match response contains {productId: #number, inventory: 0}
```

BUILT-IN KEYWORDS

VARIABLE EXTRAPOLATION

NATIVE JSON, XML, GRAPHQL

FUZZY MATCHING

EXTENDING THE DSL: JAVASCRIPT & JAVA INTEROP

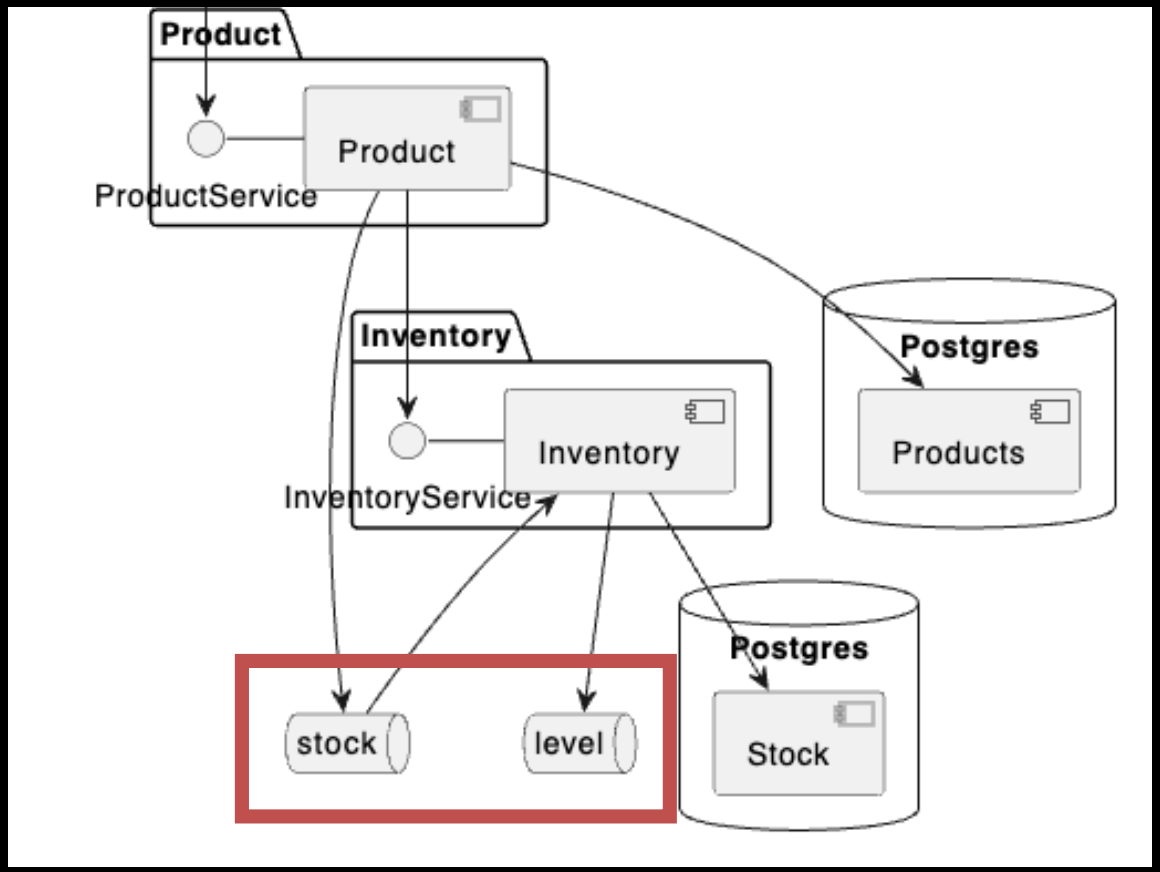
```
Background:
  * def wait = function(pause){java.lang.Thread.sleep(pause)}
  ...

Scenario: create a product
  ...
  When method post
  Then status 201
  And wait(2000)
  ...
```

DEFINE FUNCTION

USE IN SCENARIO STEP

JAVA INTEROP: EVENT-DRIVEN CONTRACTS



JAVA INTEROP: EVENT-DRIVEN CONTRACTS

```
public interface EventProducer {
  void produce(Object event);
}
```

```
public class QueueProducer implements EventProducer {
  ...
}
```

JMS API

```
function jms(connectionFactory) {
  return {
    "producer": function(destination) {
      var QueueProducer = Java.type('util.event.jms.QueueProducer');
      return new QueueProducer(connectionFactory, destination);
    },
    ...
  }
}
```

JMS-BASED PRODUCER FACTORY

EVENT-DRIVEN CONTRACTS

```
Background:
  * def eventBus = call read('classpath:jms.js', activemq);
  * def stockChannel = eventBus.producer('stock')

  ...

Scenario: Replenish should update stock level
  Given string inventoryEvent = {articleId: 'a101', stock: 100}
  When stockChannel.produce(inventoryEvent)
  Then ...
```

DEFINE EVENT BUS

DEFINE CHANNEL PRODUCER

EMIT EVENT

ROBUSTNESS & REPEATABILITY: MANAGING TEST DATA

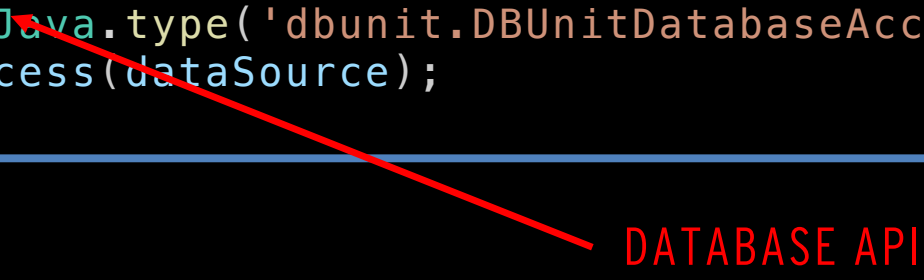
```
public interface DatabaseAccess {

  void restoreState(Map<String, Object> dataset);

}
```

```
public class DBUnitDatabaseAccess implements DatabaseAccess {
  ...
}
```

```
function databaseAccess(dataSource) {
  var DatabaseAccess = Java.type('dbunit.DBUnitDatabaseAccess');
  return new DatabaseAccess(dataSource);
}
```



MANAGING TEST DATA

```
Feature: Product system tests

  Background:
    * def productDb = call('classpath:databaseAccess.js', dataSource);
    * def initialProducts = read('classpath:/data/initialProducts.csv')
    * productDb.restoreState({product: initialProducts});

  ...
```

ESTABLISH DATABASE ACCESS

DEFINE INITIAL DATASET

RESTORE SYSTEM TO KNOWN, INITIAL STATE

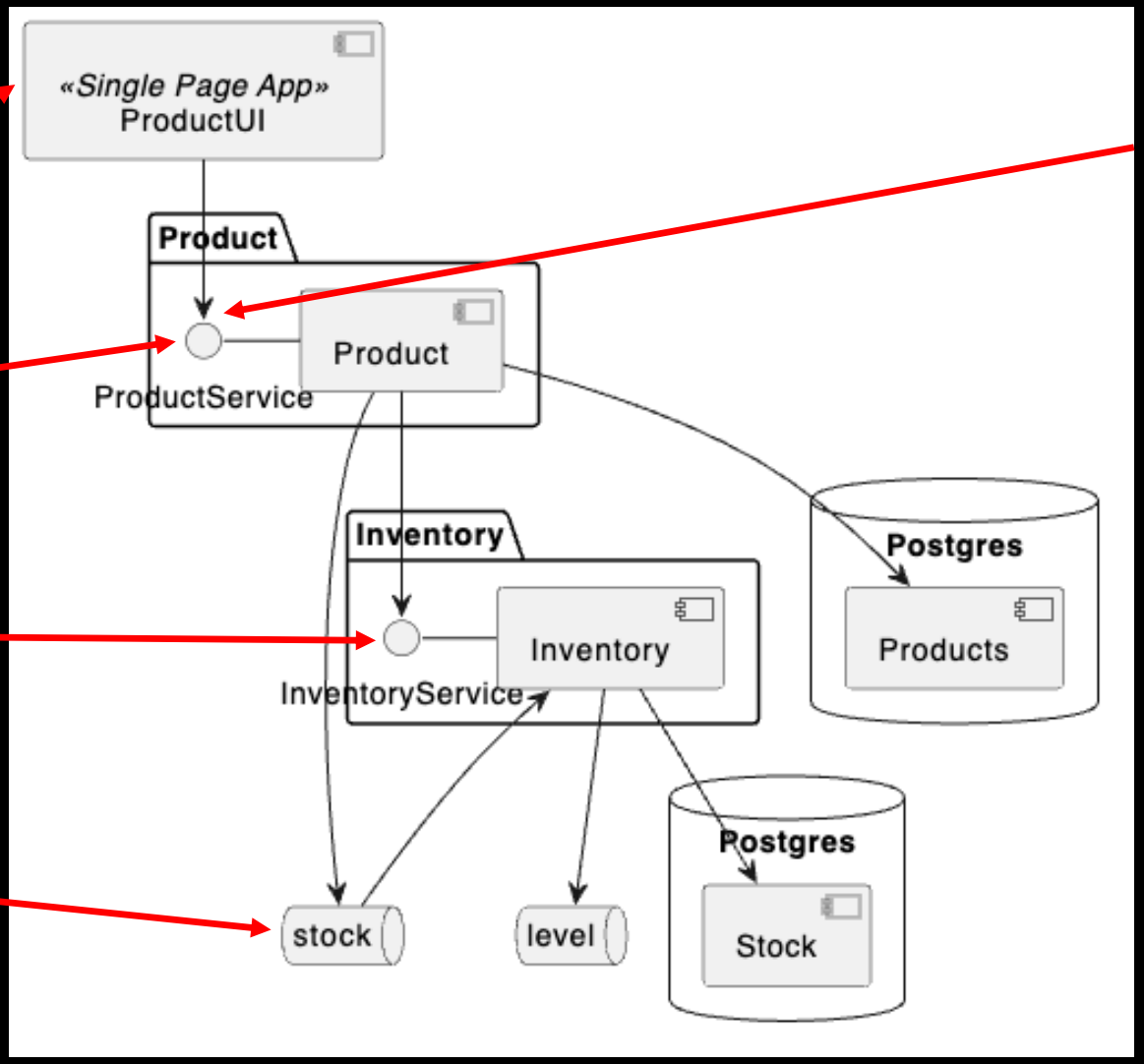
HYBRID TESTS: USER JOURNEY SYSTEM TEST

1: CREATE PRODUCT USING UI

2: RETRIEVE CREATED PRODUCT USING PRODUCT API

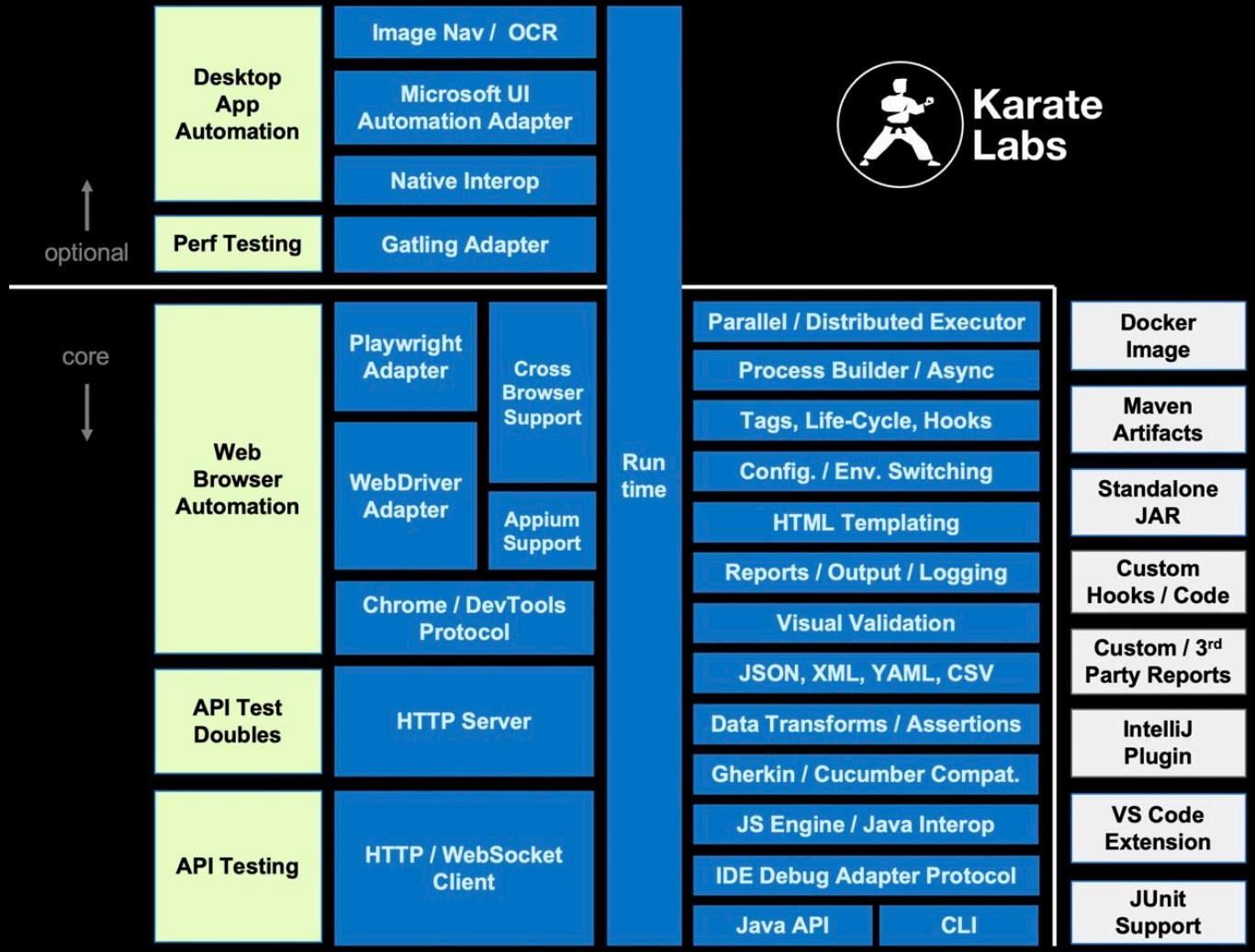
3: VERIFY INITIAL STOCK USING INVENTORY API

4: REPLENISH STOCK USING INVENTORY EVENT CHANNEL

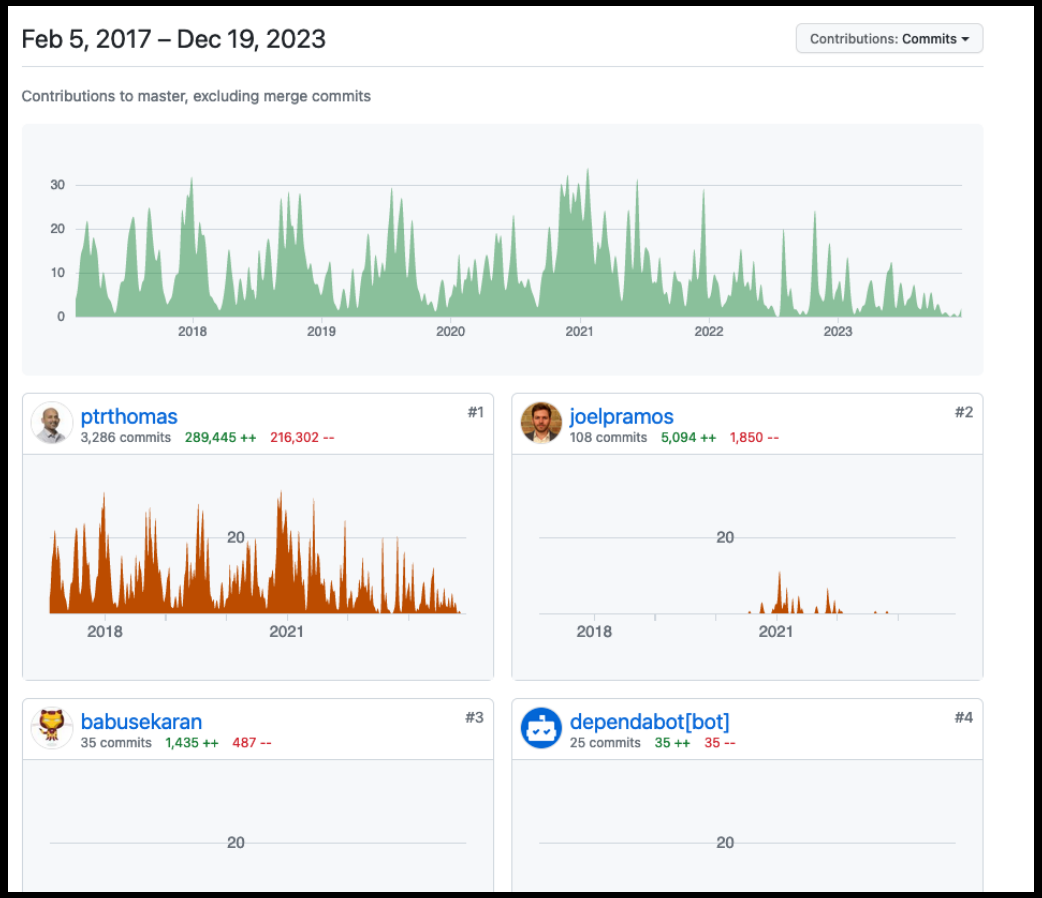


5: VERIFY UPDATED STOCK LEVEL USING PRODUCT API

THE STATE OF KARATE

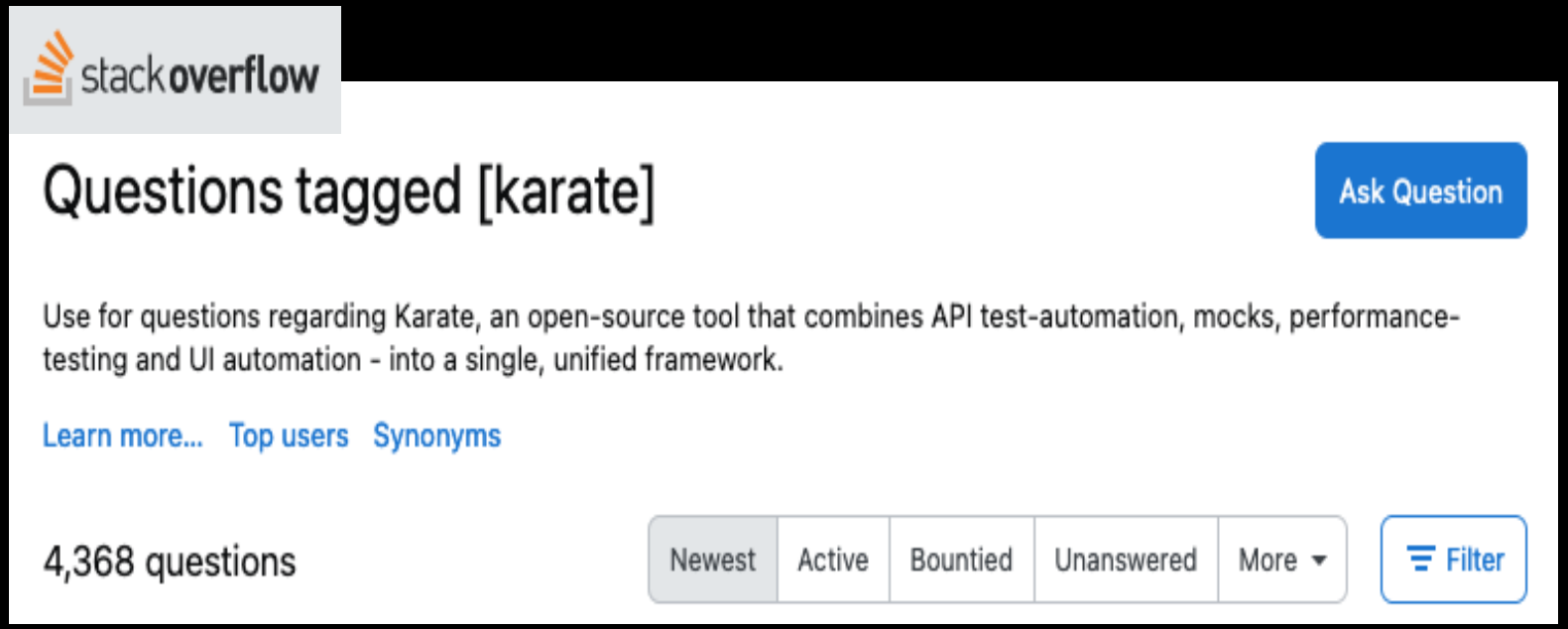


THE STATE OF KARATE



CALLISTA

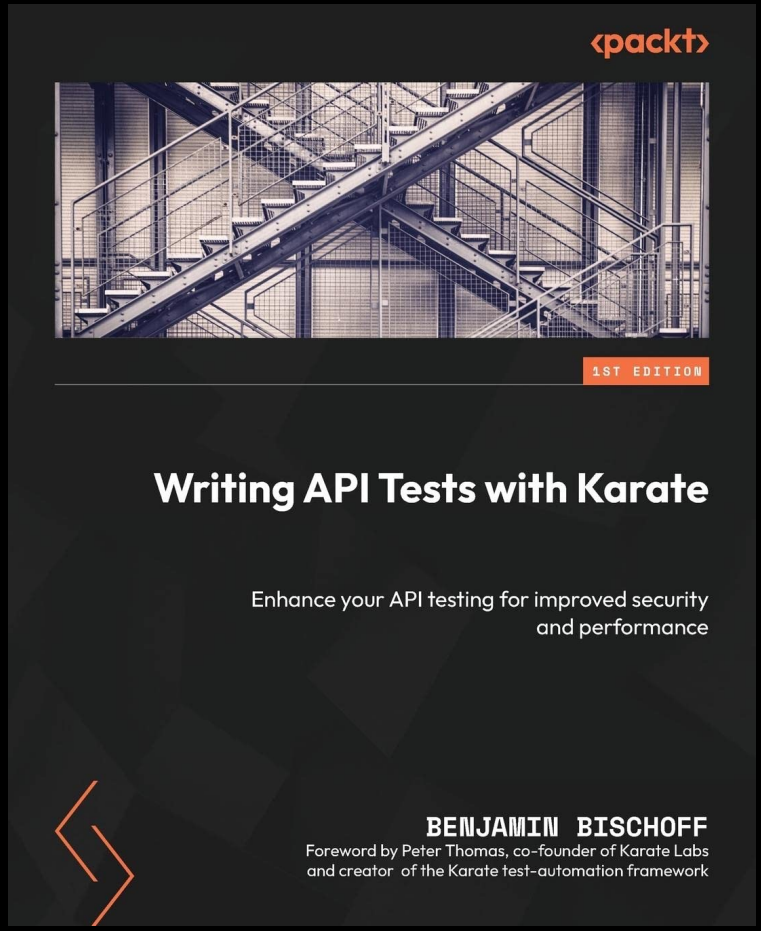
THE STATE OF KARATE



THE STATE OF KARATE

Index

Start	Maven | Gradle | Quickstart | Standalone Executable | Naming Conventions | Script Structure
Run	JUnit 5 | Command Line | IDE Support | Tags / Grouping | Parallel Execution | Java API | jbang
Report	Configuration | Environment Switching | Reports | JUnit HTML Report | Dry Run | Report Verbosity | Logging | Log Masking
Types	JSON | XML | JavaScript Functions | Reading Files | Type / String Conversion | Floats and Integers | Embedded Expressions | JsonPath | XPath | Karate Expressions
Variables	`def` | `text` | `table` | `yaml` | `csv` | `string` | `json` | `xml` | `xmlstring` | `bytes` | `copy`
Actions	`assert` | `print` | `replace` | `get` | `set` | `remove` | `configure` | `call` | `callonce` | `eval` | `listen` | `doc` | `read()` | `compareImage` | `karate` JS API
HTTP	`url` | `path` | `request` | `method` | `status` | `soap action` | `retry until`
Request	`param` | `header` | `cookie` | `form field` | `multipart file` | `multipart field` | `multipart entity` | `params` | `headers` | `cookies` | `form fields` | `multipart files` | `multipart fields`
Response	`response` | `responseBytes` | `responseStatus` | `responseHeaders` | `responseCookies` | `responseTime` | `responseType` | `requestTimeStamp`
Assert	`match ==` | `match !=` | `match contains` | `match contains only` | `match contains any` | `match contains deep` | `match contains only deep` | `match !contains` | `match each` | `match each contains deep` | `match header` | Fuzzy Matching | Schema Validation | `contains` short-cuts
Re-Use	Calling Other `*.feature` Files | Data Driven Features | Calling JavaScript Functions | Calling Java Code | Commonly Needed Utilities | Data Driven Scenarios
Advanced	Polling | Conditional Logic | Before / After Hooks | JSON Transforms | Loops | HTTP Basic Auth | Header Manipulation | GraphQL | Websockets / Async | `call` vs `read()`
More	Test Doubles | Performance Testing | UI Testing | Desktop Automation | VS Code / Debug | Karate vs REST-assured | Karate vs Cucumber | Examples and Demos



CONCLUSIONS

- Karate is a “best-of-breed” test automation framework
  - Builds on the tradition of Fit, Cucumber, Spock, RestAssured
  - Nice balance between readability and expressiveness
  - Full access to the JVM Eco System
  - Covers much ground, which allows for powerful “hybrid” testing

- BDD purists may dislike the bias towards expressivity (which may arguably harm business readability)
  - Javascript is not everyone’s cup of tea
- Somewhat unclear future direction Karate Open Source vs the KarateLabs commercial offering?

HTTPS://ICON-LIBRARY.COM/ICON/PROS-AND-CONS-ICON-18.HTML.HTML

THANK YOU!

BJÖRN BESKOW

CADEC 2024-01-18 & 2024.01.24 | CALLISTAENTERPRISE.SE

CALLISTA