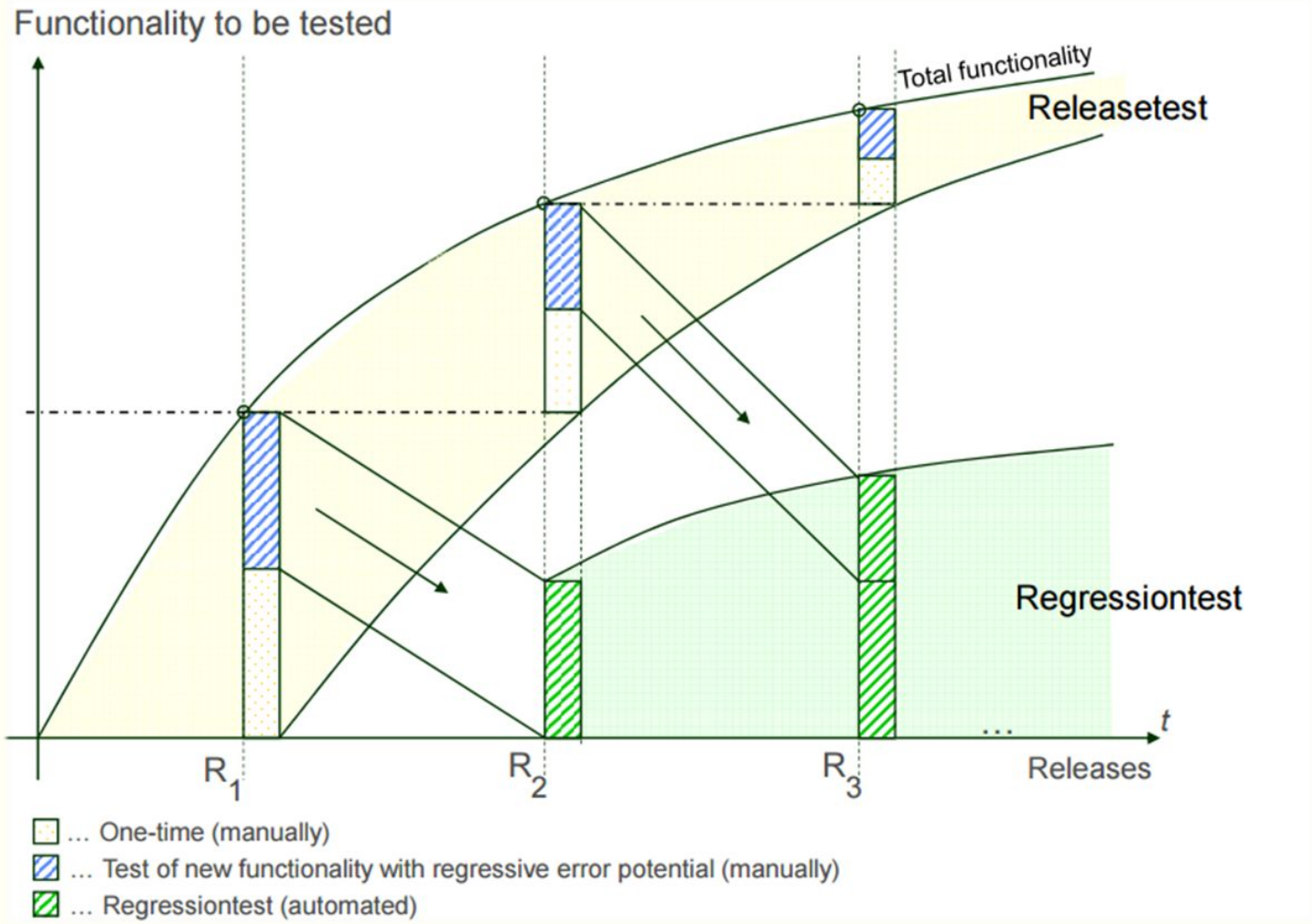
IV. AUTOMATIZACIÓN DE PRUEBAS Y DEVOPS

“It is automation, not automagic.”

IV – AUTOMATIZACIÓN DE PRUEBAS

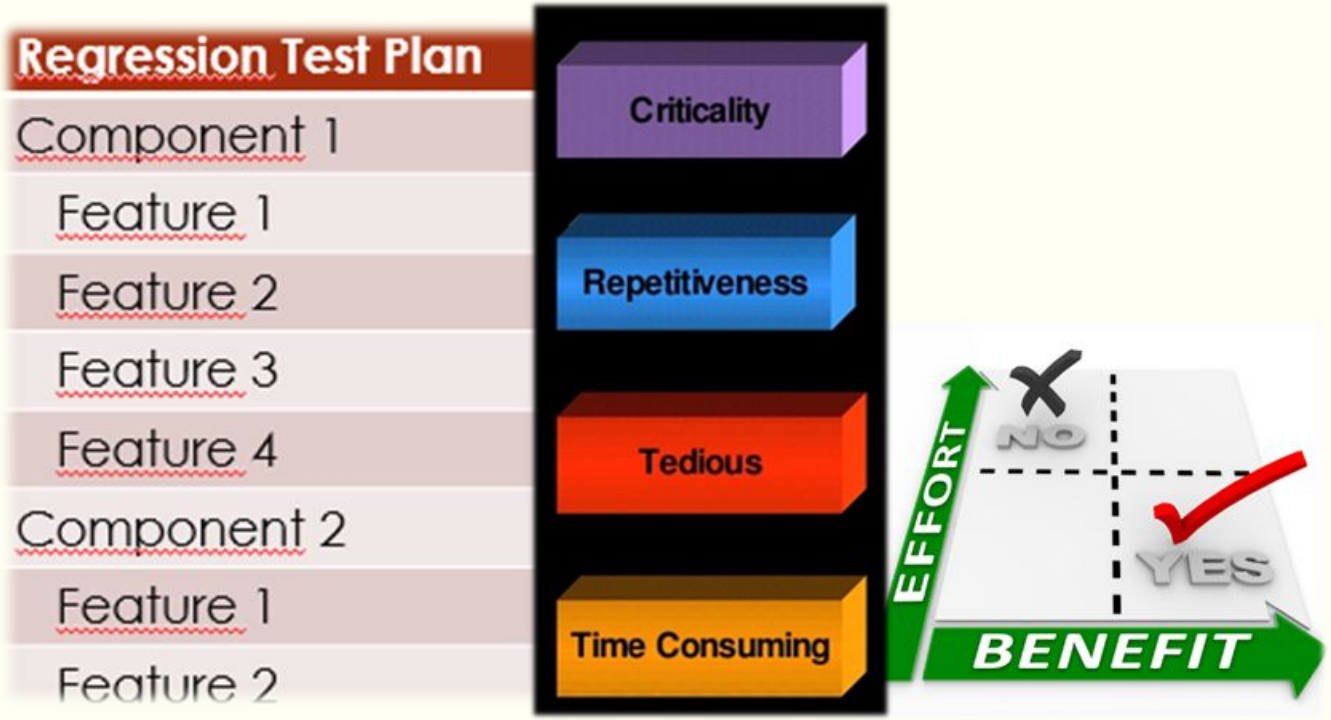
La automatización es la tecnología que permite que un proceso o procedimiento sea ejecutado con la mínima asistencia humana.

Fail-fast Principle: El principio *fail-fast* nos indica que debemos fallar rápido y pronto: Si ocurre un error, algo inusual o inesperado, esto debe ser detectado inmediatamente.



IV – AUTOMATIZACIÓN DE PRUEBAS

Seleccionar las funcionalidades correctas para automatizar determina a la larga el éxito de la automatización.



Pruebas → **Factores** → **Criterio**

IV – AUTOMATIZACIÓN DE PRUEBAS

4.1 *Frameworks* de Automatización

Es un conjunto de herramientas y librerías que proveen a sus usuarios beneficios que les ayudan a desarrollar, ejecutar y generar reports de casos de prueba de forma eficiente.

Es un sistema integrado que establece unas reglas de automatización de pruebas para un producto. Integra librerías de funciones, fuentes de datos y varios módulos reutilizables. Provee las bases de la automatización y simplifica el esfuerzo.

Además controla la ejecución de las pruebas de forma parametrizada y los reportes, pudiendo ejecutar diferentes tipos de pruebas en diferentes entornos bajo demanda.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.1 *Frameworks* de Automatización

IV – AUTOMATIZACIÓN DE PRUEBAS

4.1 *Frameworks* de Automatización

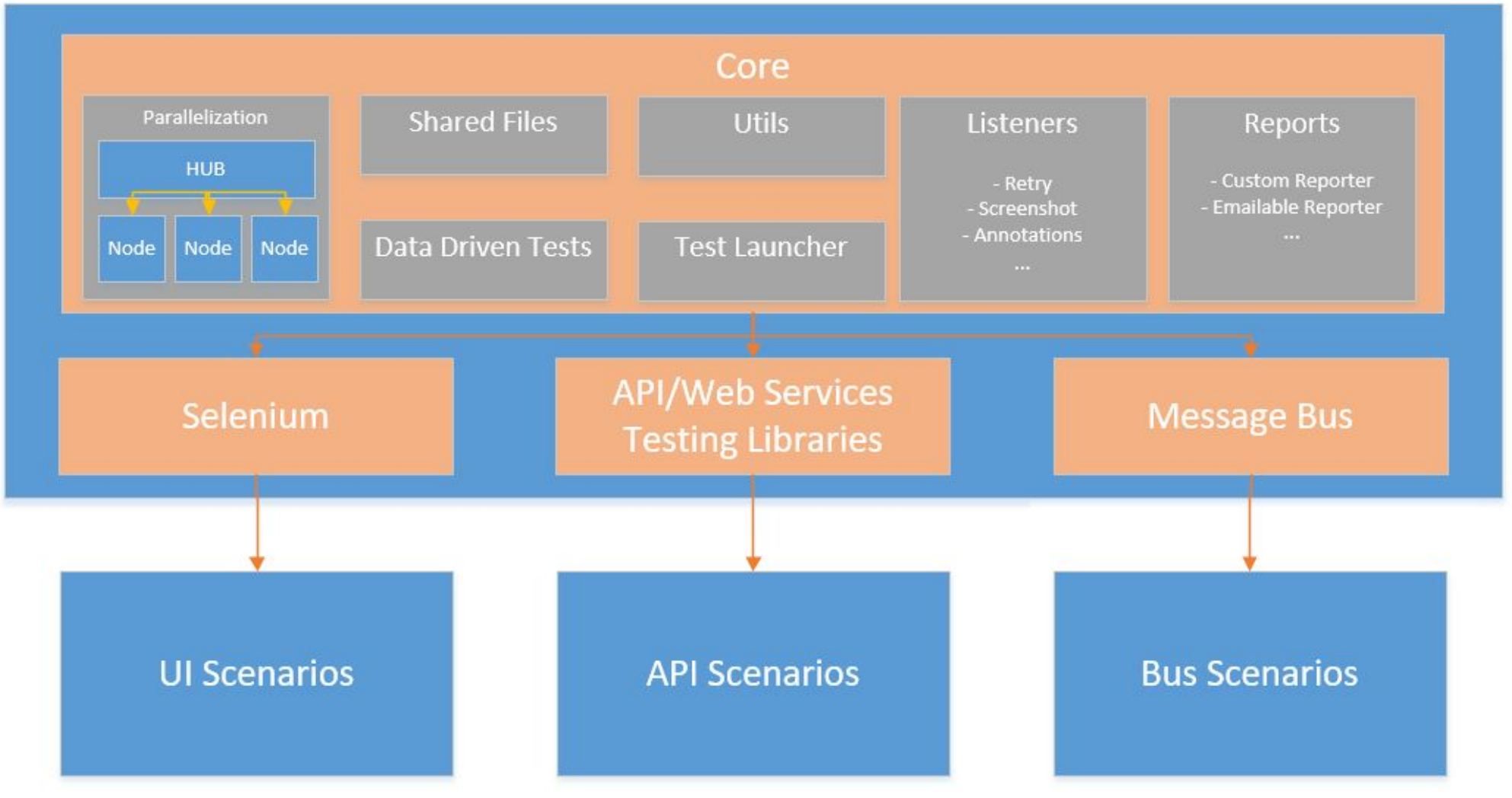
Problema 1: Expectativas
Con frecuencia, las herramientas se compran con la expectativa de que la adquisición en sí misma logra la automatización, por lo que se produce la decepción cuando los resultados no se reciben de inmediato: cada organización y aplicación es diferente.

Problema 2: Dinero
Adquirir una herramienta implica gastar dinero para software, aunque la herramienta en sí misma puede anunciarse como "fácil de usar", esto es diferente de "fácil de implementar". Es probable necesitar formación y quizás consultoría.

Problema 3: Habilidades
La herramienta requiere el tipo correcto de recursos o puedes encontrarte con una herramienta y nadie para implementarla.
Tener una herramienta no significa que puedes sobrevivir con menos habilidades o experiencia: es exactamente lo contrario.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.1 *Frameworks* de Automatización



IV – AUTOMATIZACIÓN DE PRUEBAS

4.3 Pruebas Unitarias y TDD

Las pruebas unitarias o *Unit testing*, forman parte de los diferentes procedimientos que se pueden llevar a cabo dentro de la metodología ágil. Son principalmente trozos de código diseñados para comprobar que el código principal está funcionando como esperábamos. Pequeños test creados específicamente para cubrir todos los requisitos del código y verificar sus resultados.

El proceso que se lleva a cabo, consta de tres partes.

- El ***Arrange***, donde se definen los requisitos que debe cumplir el código principal.
- El ***Act*,** el proceso de creación, donde vamos acumulando los resultados que analizaremos.
- Y el ***Assert***, que se considera el momento en que comprobamos si los resultados agrupados son correctos o incorrectos.

Dependiendo del resultado, se valida y continúa, o se repara, de forma que el error desaparezca.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.3 Pruebas Unitarias y TDD

- **Automatizable**; Aunque los resultados deben ser específicos de cada test unitario desarrollado, los resultados se pueden automatizar, de forma que podemos hacer las pruebas de forma individual o en grupos.

- **Completas**; El proceso consta de pequeños test sobre parte del código, pero al final, se debe comprobar su totalidad.

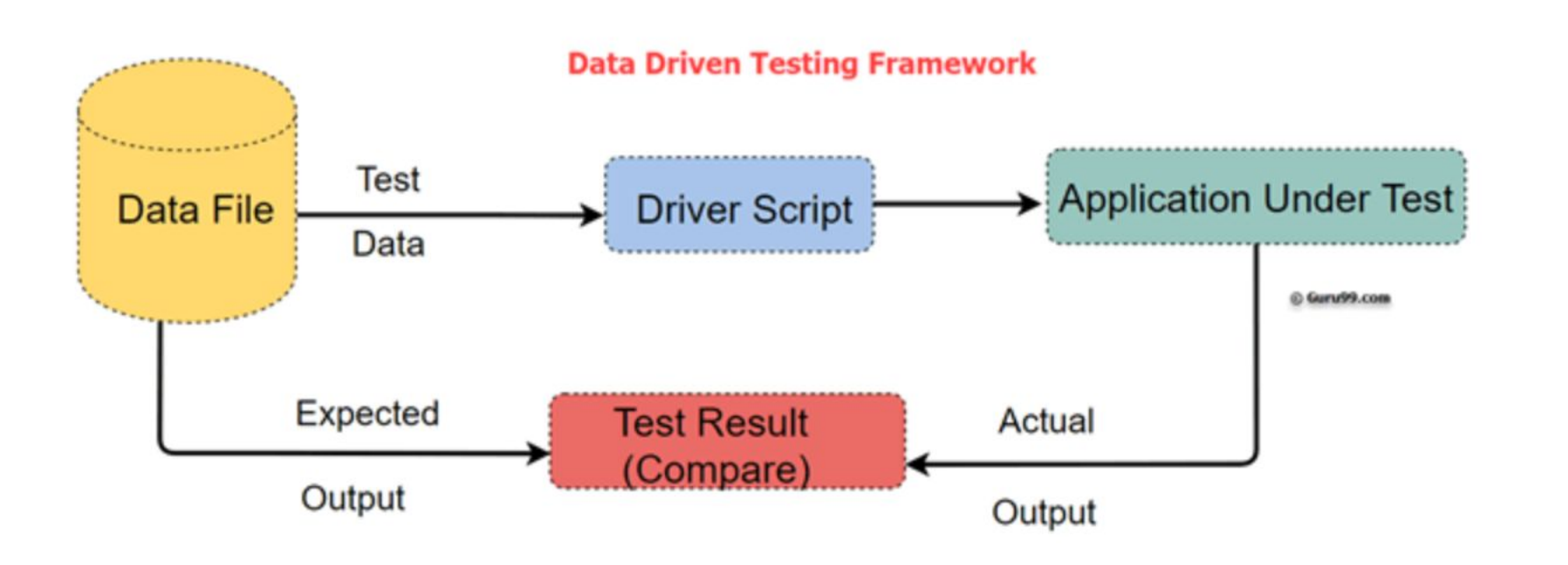
- **Repetibles**; En el caso de repetir las pruebas de forma individual o grupal, el resultado debe ser siempre el mismo dando igual el orden en que se realicen los test, los *tests* se almacenan para poder realizar estas repeticiones o poder usarlos en otras ocasiones.

- **Independientes**; Es un código aislado que se ha creado con la misión de comprobar otro código muy concreto, no interfiere en el trabajo de otros desarrolladores.

- **Rápidos de crear**; a pesar de lo que muchos desarrolladores opinen, el código de los *tests* unitarios no debe llevar más de 5 minutos en ser creado, están diseñados para hacer que el trabajo sea más rápido..

IV – AUTOMATIZACIÓN DE PRUEBAS

4.3 Pruebas Unitarias: DDT (*Data Driven Testing*)



Data Driven Testing is important because testers frequently have multiple data sets for a single test and creating individual tests for each data set can be time-consuming. Data driven testing helps keeping data separate from test scripts and the same test scripts can be executed for different combinations of input test data and test results can be generated efficiently.

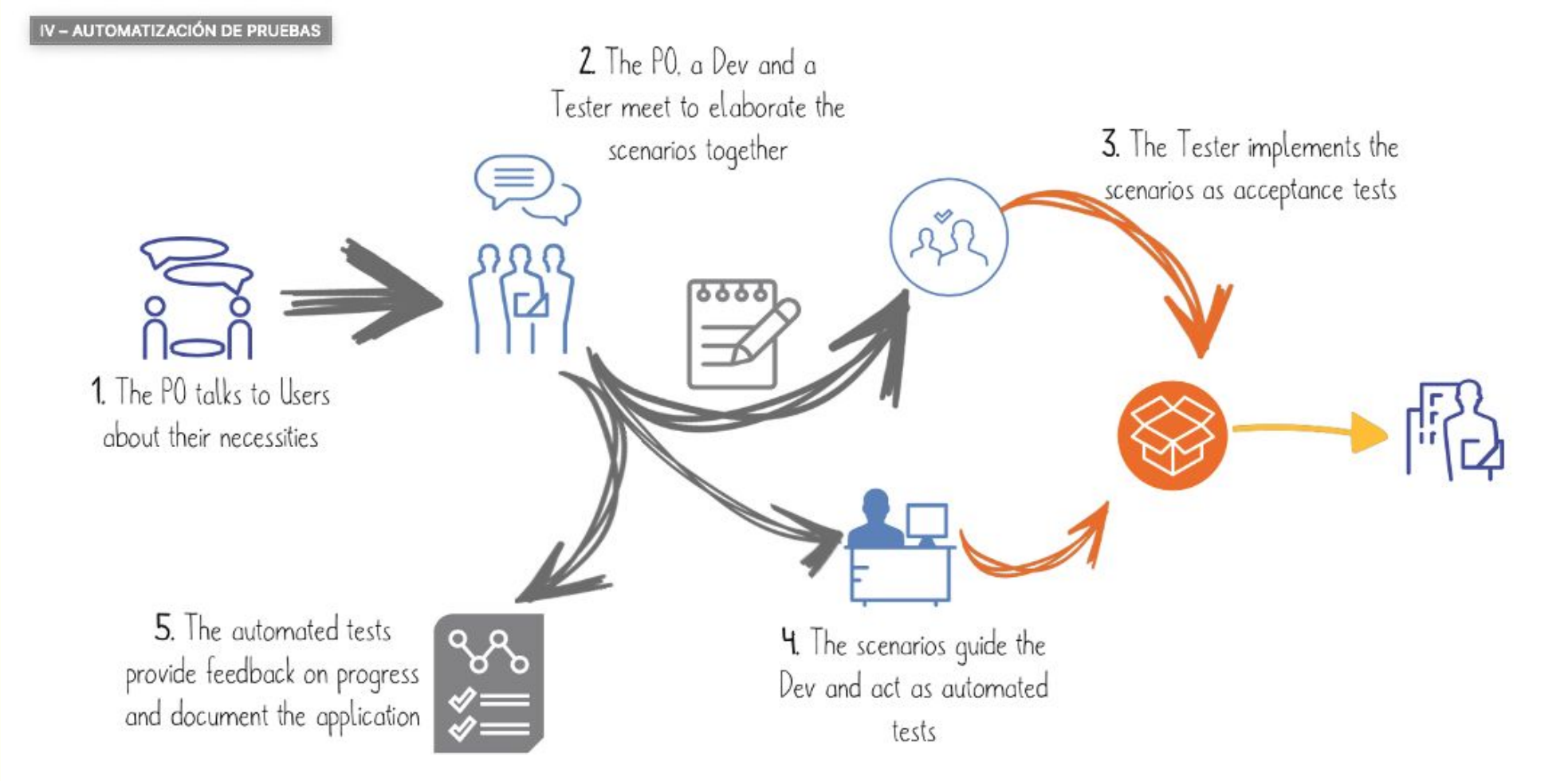
IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

```
[TestClass, System.Runtime.InteropServices.GuidAttribute( )]
public class EnquiryTests : OriginTestBase
{
    [Ignore]
    [TestMethod,
    TestCategory(_classCategoryName), TestCategory(TestCategories.Validation), TestCategory(TestCategories.OriginSpecificTest)]
    public void Enquiry_HappyPathAndCheckAllDocumentsHaveBeenGenerated()
    {
        RunTest(() =>
        {
            var emlDocument = "eml";

            string newPolicyRef = .CreatePolicy(EnquiryExcelData);
            Assert.IsTrue(ManagerFactory.QuoteComplete.IsPolicyRefVisible(), "Policy Ref is not shown");
            Assert.AreEqual(newPolicyRef, ManagerFactory.QuoteComplete.GetPolicyRef(), "Created is " + newPolicyRef + ", displayed" + ManagerFactory.QuoteComplete. );

            //upload document to enquiry
            ManagerFactory.InitialDataCapture.SelectDocumentsTab();
            string enquiryDoc = ManagerFactory.Document.UploadTempDocument();
            TestHelper.IsTrue(ManagerFactory.Document.ValidateLastUpload(enquiryDoc), "Document not correctly uploaded ");
            File.Delete(System.IO.Path.GetTempPath() + enquiryDoc);
            var pol = Repository.Instance.Policy.GetPolicyByRef(newPolicyRef);

            string transactionConfirmationDocument = "(" + pol.policyRef + ")";

            ManagerFactory.Policy.Goto(pol.policyId);
            ManagerFactory.Policy.SelectTab(PolicyPageTabs.Documents);
            Assert.IsTrue(ManagerFactory.Document.DocumentNameExists(enquiryDoc), "Document '" + enquiryDoc + "' not generated");
            Assert.IsTrue(ManagerFactory.Document.DocumentNameExists(emlDocument), "Document '" + emlDocument + "' not generated");

        }, _classCategoryName, GetJiraLink( ));
    }
```

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD



IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD (*Behaviour Driven Development*)

- *Behavior Driven Development* (BDD) es una herramienta que une a las partes interesadas de Producto, Desarrollo y Calidad para garantizar una comunicación efectiva.

- El resultado final es un proceso simplificado que produce mejores resultados: funcionalidades que los clientes realmente necesitan, productos de mayor calidad y menos tiempo para solucionar problema.

- Se usa el formato *Gherkin* (“*Given-When-Then*"), que está escrito en inglés simple, lo que hace que sea fácil de entender independientemente de la experiencia técnica.

- Los escenarios de *Gherkin* pueden ejecutarse fácilmente como pruebas automatizadas utilizando una herramienta como *Specflow*.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

Feature: Feature name

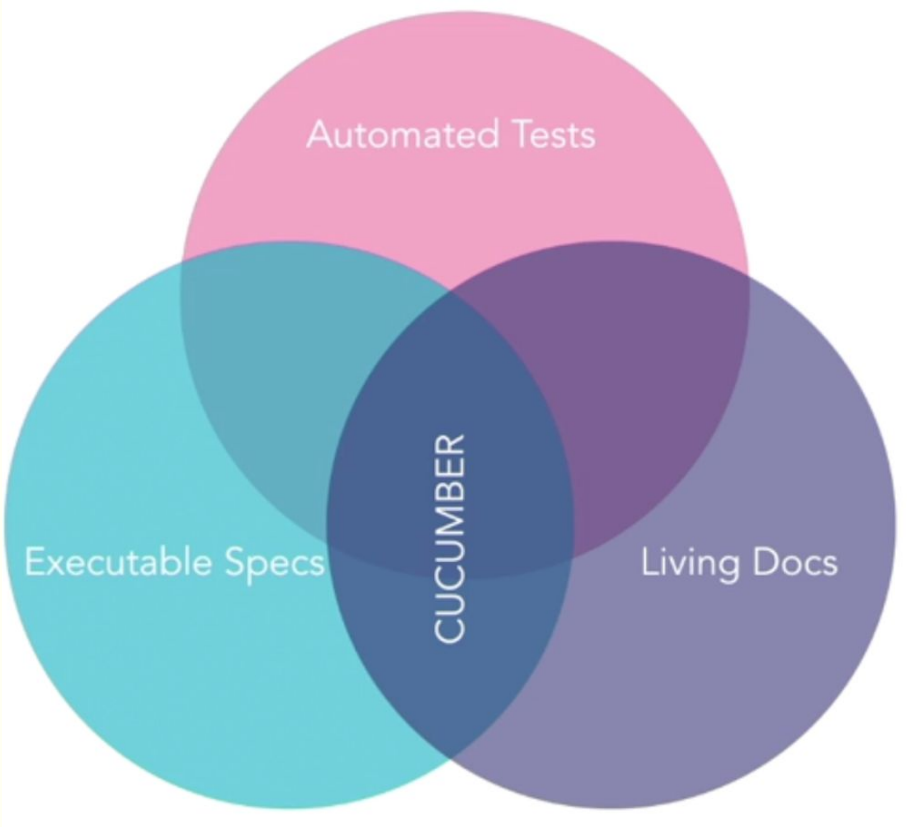
Some terse, yet descriptive, text of what is desired

Scenario: Some determinable business situation

Given some precondition

When some action by the actor

Then some testable outcome is achieved.



IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

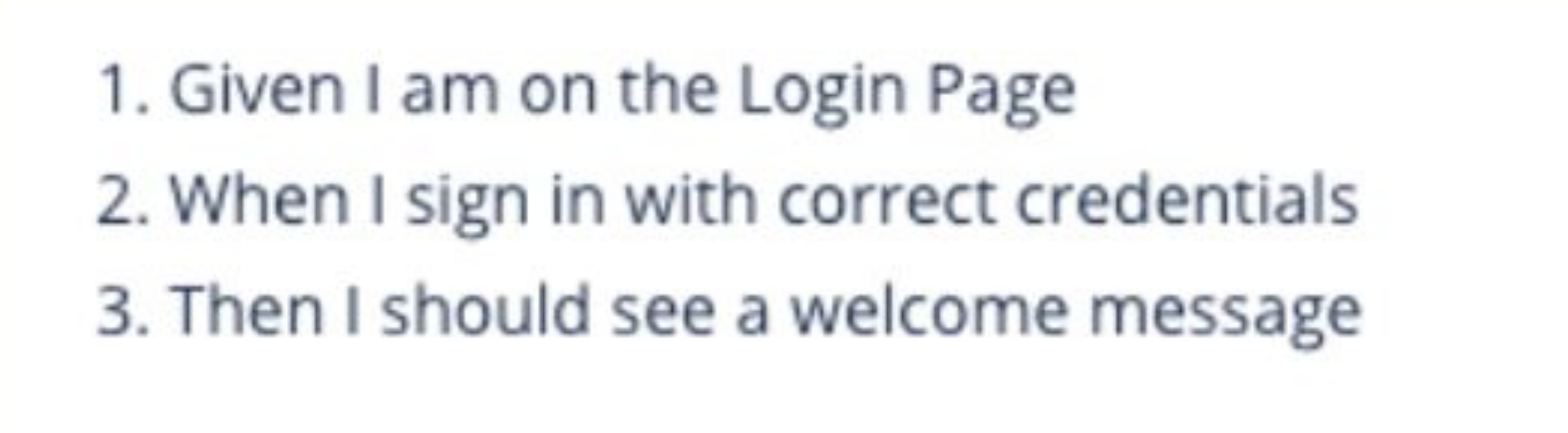
Imperative testing: Detallar todo lo necesario para hacer la prueba paso a paso.

1. Given I open a browser
2. And I navigate to `http://example.com/login`
3. When I type in the username field `bob97`
4. And I type in the password field `F1d0`
5. And I click on **Submit** button
6. Then I should see the message `Welcome Back Bob`

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

Declarative testing: Pasos a nivel de negocio, sin especificar detalles



IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

FEATURE	El propósito del FEATURE es proporcionar una descripción de alto nivel de una de las funciones de software y agrupar SCENARIOs relacionados.
SCENARIO o EXAMPLE	Un SCENARIO es un ejemplo concreto que contiene una regla de negocio. Consiste básicamente en una definición en el patrón 'Given-When-Then'.
GIVEN	GIVEN es parte de patrón 'Given-When-Then'. Se utilizan para describir el contexto inicial del sistema: la escena del escenario. El propósito de los Given es poner el sistema en un estado concreto antes de que el usuario (o sistema externo) comience a interactuar con el sistema (en los WHEN). Es importante evitar hablar sobre la interacción del usuario en este patrón.
WHEN	WHEN es la segundo requisito del patrón 'Given-When-Then'. Se utilizan para describir un evento o una acción. Puede ser una persona que interactúa con el sistema o puede ser un evento desencadenado por otro sistema.
THEN	THEN, la última descripción del patrón 'Given-When-Then'. Se utilizan para describir el resultado esperado. La definición de un THEN debe usar una aserción para comparar el resultado real (lo que el sistema realmente hace) con el resultado esperado (lo que se supone que debe hacer el sistema).
AND	AND se utiliza para añadir alguna condición más en alguno de los patrones Given, When o Then
BUT	Al igual que el AND se utiliza en los patrones Given, When o Then, pero en este caso se utiliza como condición extra.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

BACKGROUND	Ocasionalmente, te encontrarás repitiendo los mismos GIVEN en muchos SCENARIO de una FEATURE. Si es el caso, como se repite en cada escenario, esto es una indicación de que los patrones no son esenciales para describir los escenarios; Son detalles generales. Literalmente, puedes moverlos agrupándolos en un BACKGROUND.
SCENARIO OUTLINE	El SCENARIO OUTLINE se puede usar para ejecutar varios SCENARIO varias veces, con diferentes combinaciones de valores.
|	Los “Data Tables” o |, son útiles para pasar una lista de valores a una definición de patrones.
"""	Doc Strings o “”” es útil si necesitas añadir mucha información a los patrones.
@	Prefijo para una etiqueta: @. Las etiquetas pueden ser colocadas antes de los patrones o SCENARIO. El objetivo principal es ayudarte a filtrar SCENARIOs.
#	Para definir comentarios. Solo se permiten al comienzo de una nueva línea.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

```
Feature: Búsqueda en Google
  Como usuario web, quiero buscar en Google para poder responder mis dudas.

  Scenario: Búsqueda simple en Google
    Given un navegador web en la página de Google
    When se introduce la palabra de búsqueda "pingüino"
    Then se muestra el resultado de "pingüino"
    And los resultados relacionados incluyen "Pingüino emperador"
    But los resultados relacionados no incluyen "ping pong"
```

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD

Si necesitas añadir documentación extra en algún patrón, puedes utilizar las cadenas de texto “Doc Strings”

```
Feature: Búsqueda en Google
  Como usuario web, quiero buscar en Google para poder responder mis dudas.

  Scenario: Búsqueda simple en Google
    Given un navegador web en la página de Google
    When se introduce la palabra de búsqueda "pingüino"
    Then se muestra el resultado de "pingüino"
    And la página de resultados muestra el texto de Wikipedia
      """
      Nombre científico: Spheniscidae
      Clase: Aves
      """
```

IV – AUTOMATIZACIÓN DE PRUEBAS

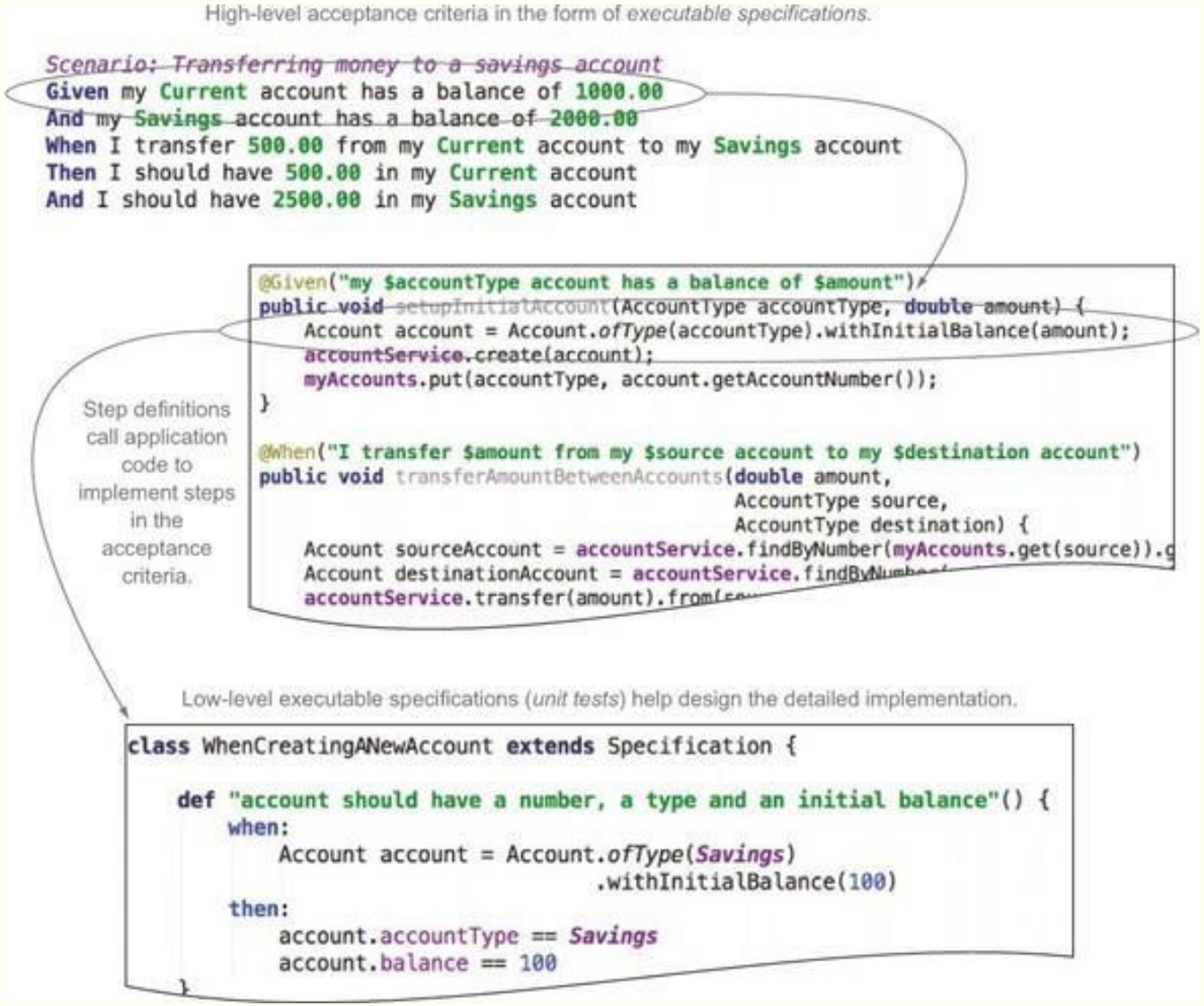
4.4 Pruebas de Aceptación y BDD

Los SCENARIO OUTLINES te permitirán automatizar aún más la definición en Gherkin. Observa el ejemplo anterior, los dos escenarios son idénticos si no fuese por sus términos de búsqueda “pingüino” y “panda”

```
Feature: Búsqueda en Google
  Como usuario web, quiero buscar en Google para poder responder mis dudas.

  Scenario Outline: Búsqueda simple en Google
    Given un navegador web en la página de Google
    When se introduce la palabra de búsqueda "<frase>"
    Then se muestra el resultado de "<frase>"
    And los resultados relacionados "<relacionado>"

    Examples: Animales
      | frase    | relacionado         |
      | pingüino | Pingüino emperador |
      | panda    | Panda gigante      |
      | elefante | Elefante Africano  |
```

IV – AUTOMATIZACIÓN DE PRUEBAS

4.4 Pruebas de Aceptación y BDD



IV – AUTOMATIZACIÓN DE PRUEBAS

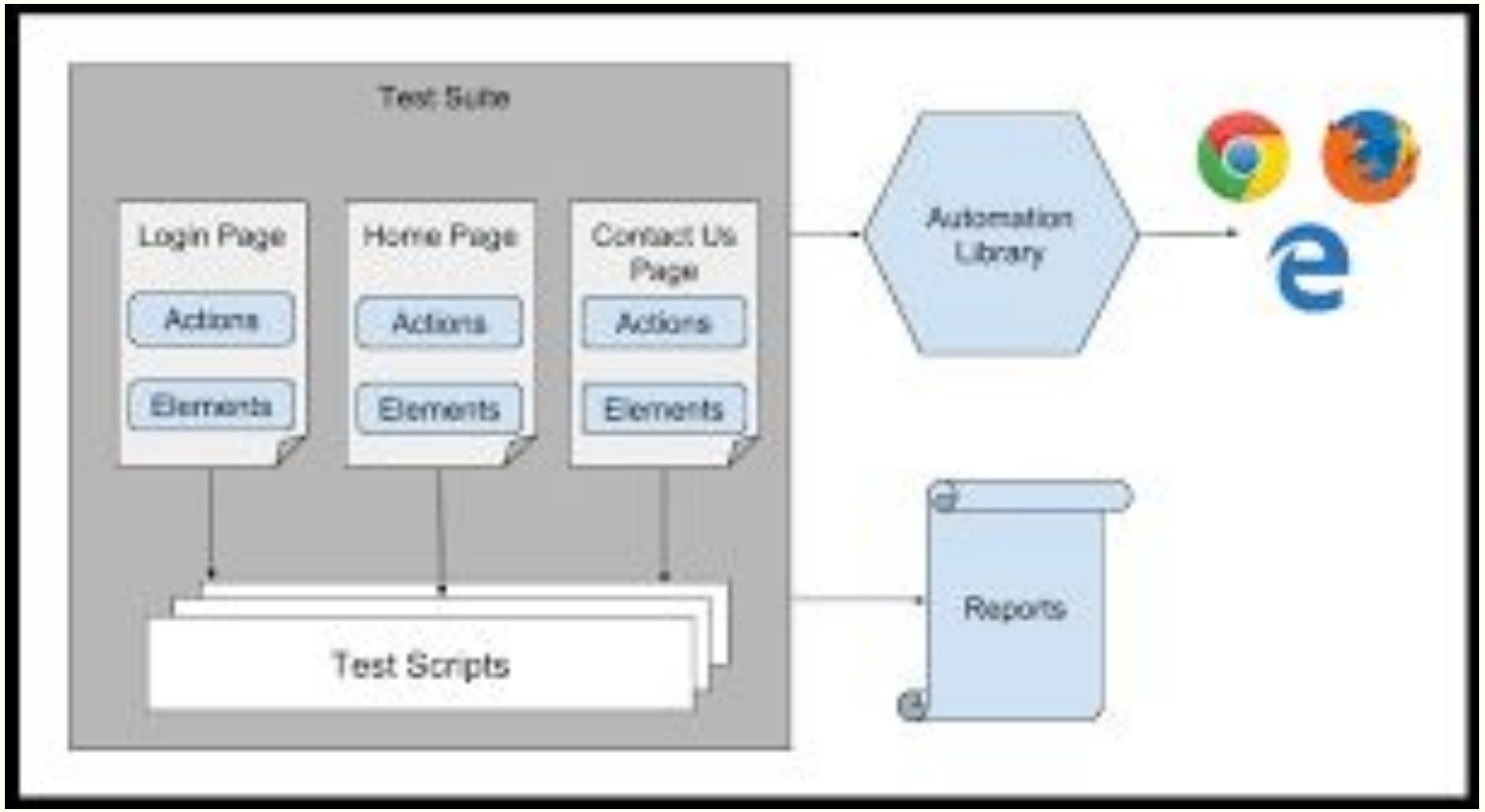
A **page object** is an object representing a Web page or component.

- It has *locators* for finding elements on the page.
- It has *interaction methods* that interact with the page under test.

Each Web page or component under test should have a page object class.

- Page objects encapsulate low-level Selenium WebDriver calls.
- That way, tests can make short, readable calls instead of complicated ones.

IV – AUTOMATIZACIÓN DE PRUEBAS



IV – AUTOMATIZACIÓN DE PRUEBAS

An *element* is a “thing” on a Web page, like a button, label, or text input.

Tests interact with elements in three steps:

1. Wait for the target element to appear
2. Get an object representing the target element
3. Send commands to the element object

IV – AUTOMATIZACIÓN DE PRUEBAS

4.5 API
Testing

Postman es un cliente HTTP que nos permite gestionar las peticiones a nuestras API´s. Postman tiene muchas funcionalidades para gestionar todo el ciclo de vida de nuestra API, como crear tests y automatizarlos para nuestras colecciones de peticiones.

- Una vez escritas las pruebas para todas nuestras peticiones y lanzando las pruebas cada vez que haya un nuevo cambio en la API, garantizamos que las aplicaciones que dependen de nuestra API funcionen como se espera. Y si las pruebas no pasan, a la vez, tendremos información sobre lo que tenemos que arreglar.

- Podemos hacer pruebas tanto como en el tipo de respuesta que damos a nuestras aplicaciones clientes, como del contenido. Cualquier cambio de esquema o de código HTTP puede ser detectado por nuestras pruebas.

- Conforme nuestra API evolucione, las pruebas evolucionarán con ella, con lo que garantizamos su mantenimiento y reducción de errores.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.5 API
Testing
https://www.postman.com/

Está compuesto por diferentes herramientas y utilidades gratuitas (en la versión free) que permiten realizar tareas diferentes dentro del mundo API REST:

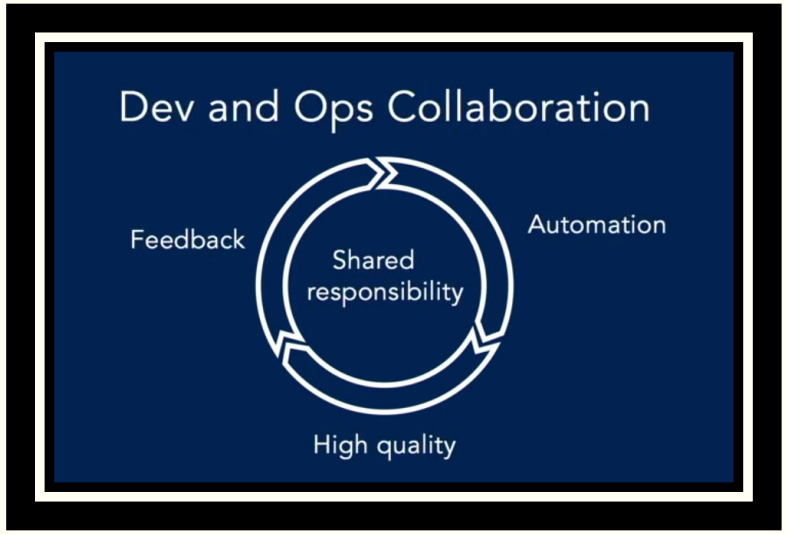
1. Creación de peticiones a APIs internas o de terceros.
2. Elaboración de tests para validar el comportamiento de APIs.
3. Posibilidad de crear entornos de trabajo diferentes (con variables globales y locales).

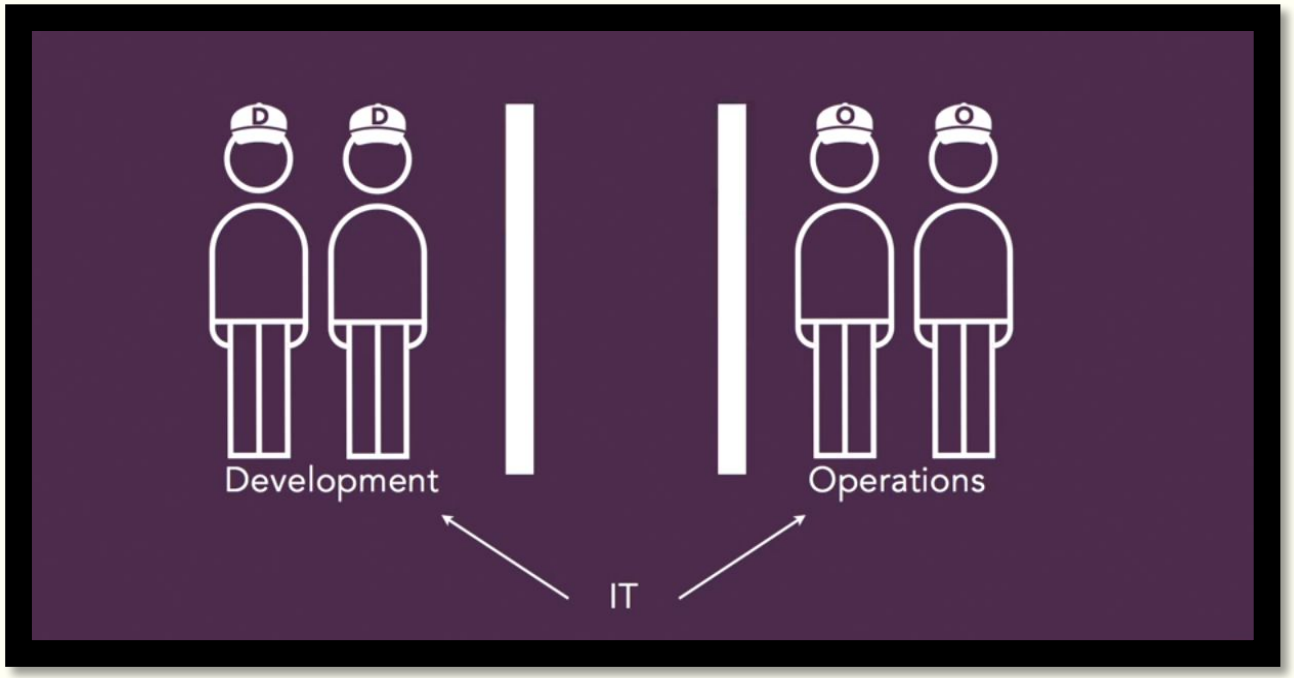
Todo ello con la posibilidad de ser compartido con otros compañeros del equipo de manera gratuita (exportación de toda esta información mediante URL en formato JSON).

IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *DevOps*

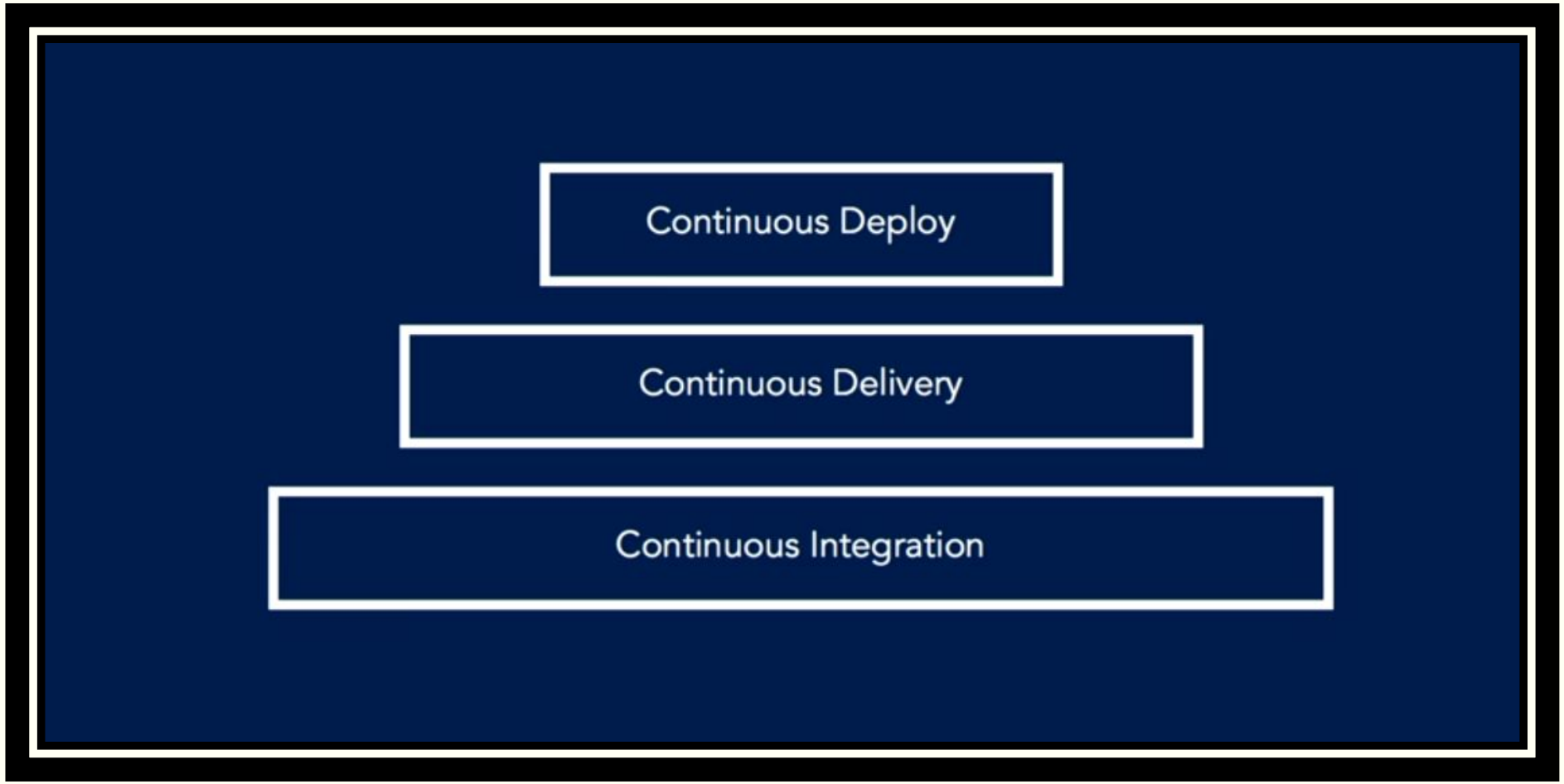
- **DevOps** = *Development* + *Operations* + *other teams*.
  - *Dev* = Código. Desde *Front-End* hasta QA.
  - *Ops* = Sistemas. Desde Linux *Administrators* hasta *Network Admins* y DBAs.
- **DevOps** != Problema Tecnológico
- **DevOps** = Problema cultural
- **KEY** = Colaboración





IV – AUTOMATIZACIÓN DE PRUEBAS

4.7 CI/CD: *DevOps*



IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *DevOps*

- ***Continuous Integration***: Es la práctica de, frecuentemente, construir y probar a nivel unitario la aplicación completa. Idealmente en cada entrada nueva de código.

- ***Continuous Delivery***: Es la práctica adicional de desplegar cada cambio a un entorno similar a producción y ejecutar pruebas automáticas de integración y aceptación.

- ***Continuous Deployment***: Extiende el concepto donde cada cambio pasa directamente por suficientes baterías de pruebas automáticas y es deplegado automáticamente en produción.

***DevOps core concept*: CI/CD**

IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Integration*

- En cada *commit*:
  - Se construye el código.
  - Se ejecutan todas las pruebas unitarias.
  - Se hacen otros pasos de validación.
  - Se empaqueta y se guarda la información del resultado.

- Si las pruebas fallan:
  - La construcción falla para todo el equipo.
  - Nuevos arreglos relanzarían todo el proceso.

- Esta disciplina promueve:
  - Bucles de *feedback* rápido.
  - Escritura de pruebas automáticas.
  - Reduce el re-trabajo.

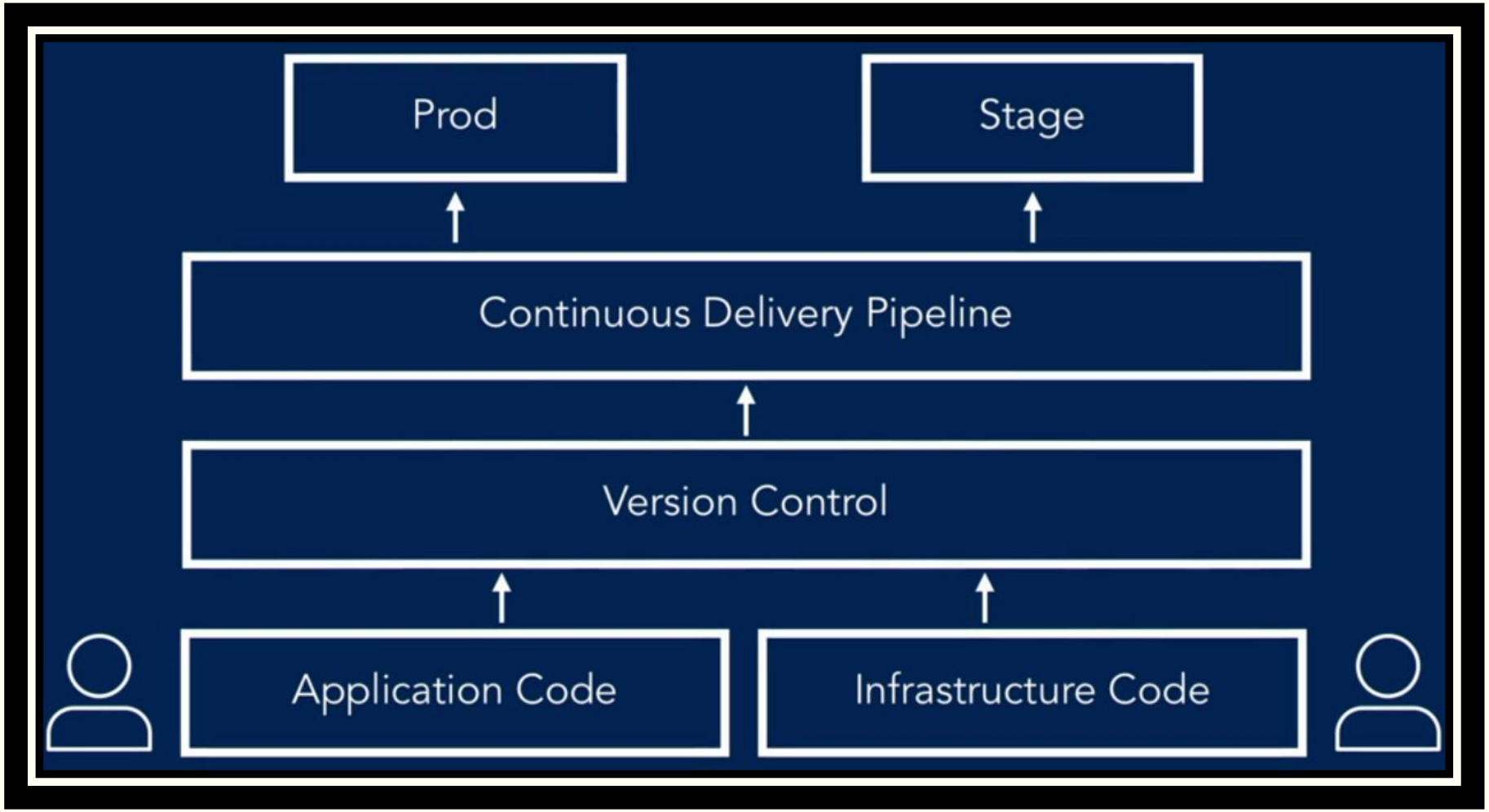
IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Delivery*

- Es una metodología *DevOps*.
- Práctica de desplegar cada cambio a un entorno similar a producción.
- Ejecutar pruebas automáticas de integración y aceptación.
- Usar entornos de prueba:
  - *Docker containers*.
  - *Virtual Machines.*
- El entorno de prueba permite:
  - Validar el proceso de despliegue.
  - Ver el estado de la funcionalidad y el rendimiento antes de un alcanzar un entorno real de Producción.
- Los artefactos construidos deben ser los mismos para todos los entornos disponibles y no ser modificados. Esto consigue una mejora en:
  - Confianza entre Dev and Ops, and QA.
  - Auditoría.

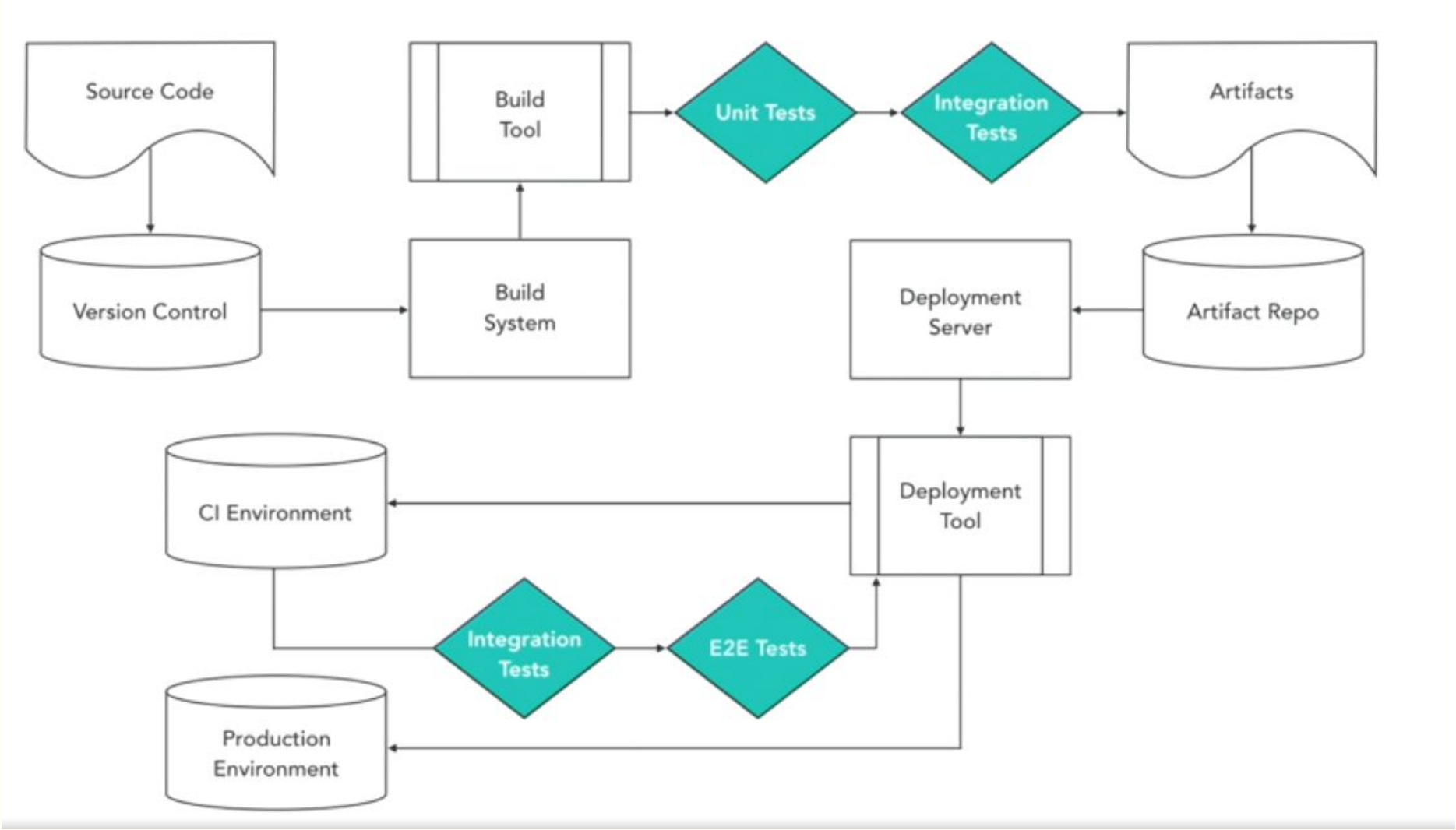
IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Delivery*



IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Pipeline*



IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Testing*

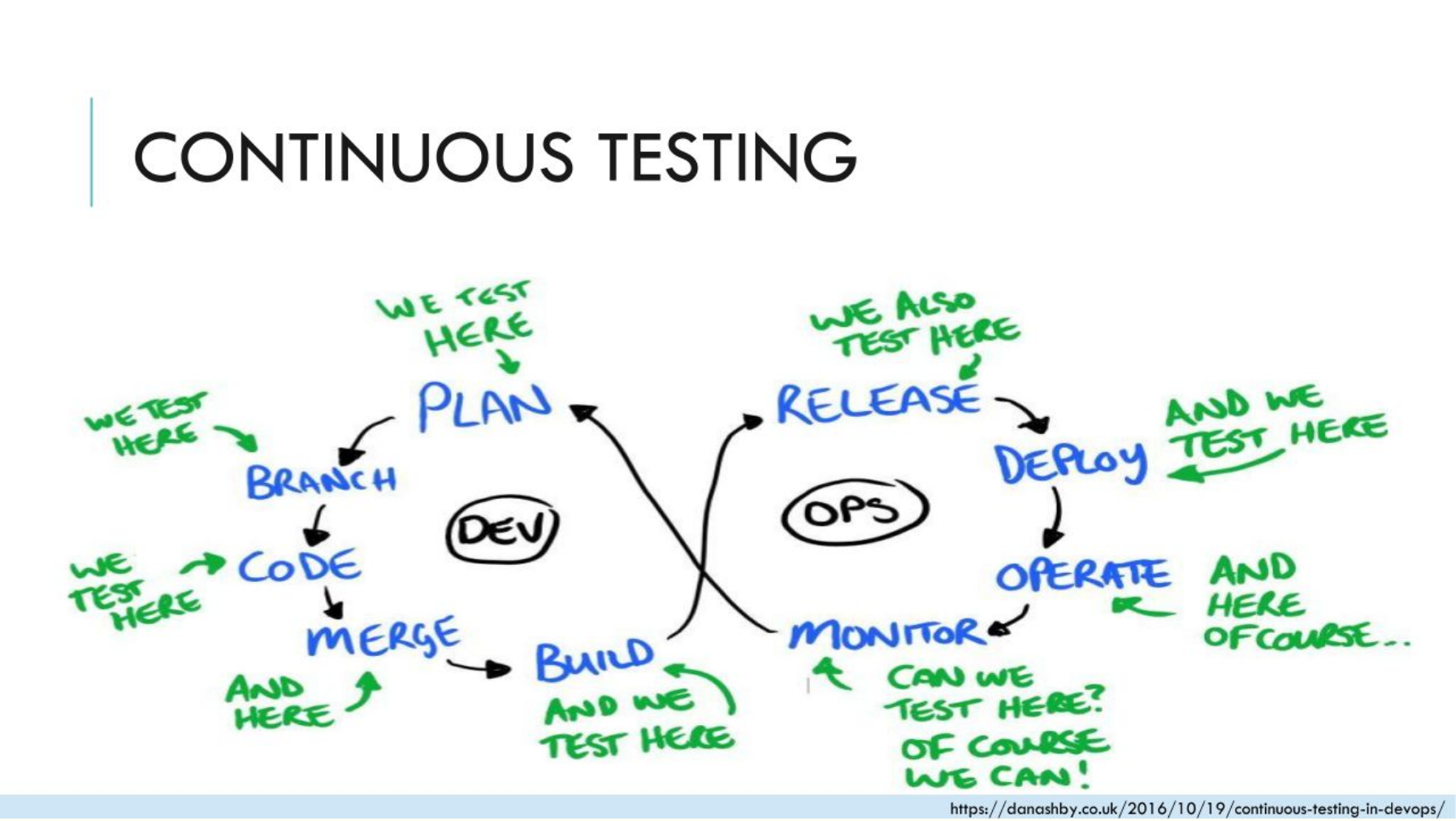
Continuous testing es el proceso que permite obtener valor analizando los riesgos asociados al negocio y la entrega de la aplicación siguiendo una aproximación CI/CD.
No importa si despliegas una vez al mes o cada minuto, Continuous Testing puede responder a la pregunta: ”¿Estamos contentos con el valor que aporta el incremento a nuestro negocio, stakeholders o usuarios?”.

“The key to building quality into our software is making sure we can get fast feedback on the impact of changes. [...] In order to build quality in to software, we need to adopt a different approach. Our goal is to run many different types of tests, both manual and automated, continually throughout the delivery process.”

“*Testing is a cross functional activity that involves the whole team, and should be done continuously from the beginning of the project.”*

IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Testing*



IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Testing*

Beneficios:

- Primeramente, obtenemos *feedback* rápido en cualquier fase, los test automáticos incluidos en nuestra *pipeline* tienen que fallar lo más pronto posible para detectar los problemas tempranamente.

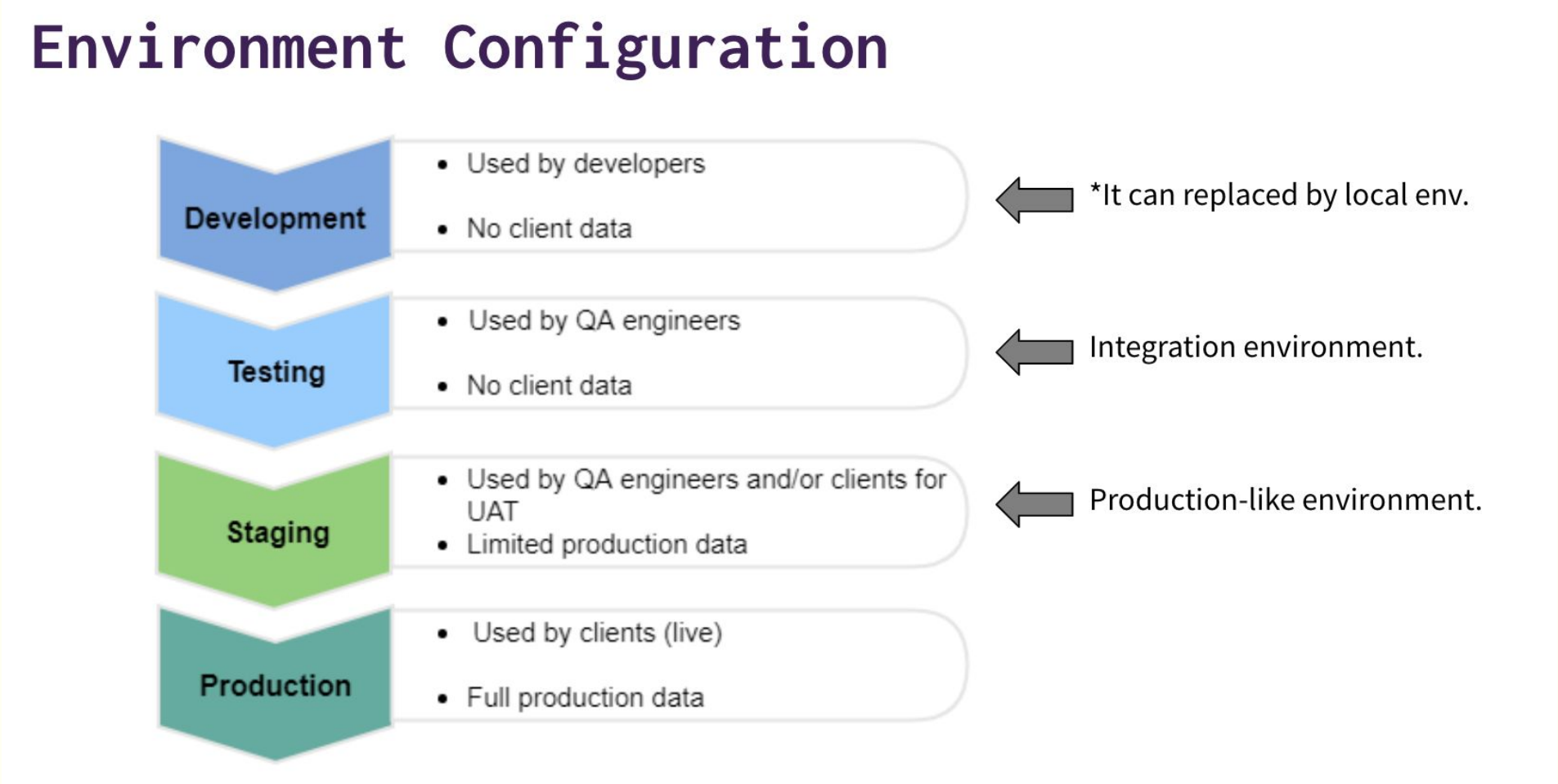
- Otra cuestión, siempre estamos aprendiendo de nuestra aplicación y la calidad que esperamos que tenga.

- Como consecuencias, acelera el desarrollo y reduce los tiempos de bloqueo, limita el trabajo en progreso beneficiándonos de entregas más temprano y mejor *time to market.*

- *Continuous Testing* incrementa la confianza de que vamos por buen camino e incrementa la moral del equipo.

IV – AUTOMATIZACIÓN DE PRUEBAS

4.6 CI/CD: *Continuous Testing*



Fuentes

- https://www.itdo.com/blog/ejemplos-bdd-behavior-driven-development-con-gherkin/
- https://apiumhub.com/es/tech-blog-barcelona/beneficios-de-las-pruebas-unitarias/
- https://testsigma.com/data-driven-testing
- https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/
- https://dzone.com/articles/testing-in-ci
- https://docs.gitlab.com/ee/ci/examples/
- https://docs.gitlab.com/ee/ci/pipelines/
- https://dev.to/ykyuen/a-simple-gitlab-ci-example-doe
- https://medium.com/@cesiztel/c%C3%B3mo-se-hace-api-testing-con-postman-978a521552f4
- https://www.paradigmadigital.com/dev/postman-gestiona-construye-tus-apis-rapidamente/