**1 - . Introducción**

La automatización de pruebas es la práctica de ejecutar pruebas de una manera repetible, que permita utilizar la información y los resultados obtenidos para:

* Mejorar la calidad del software.
* Maximizar el retorno de la inversión (ROI).

**2 -. Ventajas**

* Mejorar la eficiencia de las pruebas.
* Proporcionar una cobertura de pruebas más amplia.
* Reducir costo total de las pruebas.
* Acortar el periodo de ejecución de la prueba.
* Aumentar la frecuencia de la prueba y reducir tiempo requerido para los ciclos de prueba.
* Se pueden ejecutar mas pruebas por compilación/liberación.
* La posibilidad de crear pruebas que no se pueden realizar manualmente.
* Las pruebas están menos sujetas a errores del operador.

**2.1 -. Desventajas**

Las desventajas que nos ofrece la automatización de pruebas son las siguientes:

* Costos adicionales en herramientas, soluciones o profesionales.
* Requiere de conocimientos de programación, lo que implica que sea más difícil de solucionar errores o debuggear.
* Requiere un mantenimiento continuo porque el software evoluciona rápidamente.
* Es necesario agregar tecnologías adicionales en el stack de tu empresa.
* Las pruebas pueden volverse complejas.
* Distracción de los objetivos de la prueba, a veces no se evalúa correctamente lo que se debe automatizar o lo que no.
* Tiempos innecesarios en automatizar pruebas que hubieran sido resueltas más rápido de forma manual.

**3 -. Limitaciones**

* Probar la verificación de dos pasos:
  + La verificación de dos pasos (Two Factor Authentication) consiste en enviar un código de seguridad para identificar a un usuario, por lo tanto, por cuestiones de seguridad y de la complejidad que se requiere, no es posible automatizar este procedimiento.
* Probar los sistemas anti-spam:
  + El sistema de reCAPTCHA consiste en verificar que el usuario es una persona y no un robot o un programa.

Usualmente, estas técnicas son usadas por empresas que desean generar spam.

La verificación consiste en seleccionar imágenes relacionadas entre sí o un código, por lo que, por su complejidad, no es posible automatizarlo.

* Probar los correos automatizados:
  + La automatización de correos ya sea para realizar una restauración de contraseña o probar que un correo haya sido enviado correctamente, no es posible en su totalidad.

Existen muchas limitaciones para probar el envío de correos, aunque no es imposible porque hay herramientas que te permiten conectarte a un servidor y observar algunos de los resultados que se desean probar.

Sin embargo, el costo de automatizar estas pruebas puede ser elevado porque es necesario adquirir el servicio de correos.

Por el motivo anterior y el tiempo que es necesario invertir para este tipo de automatizaciones, no es rentable hacerlo.

* Probar los sensores de los dispositivos móviles:
  + Los sensores de los dispositivos móviles son difíciles de probar de manera automatizada, porque su comportamiento es difícil de emular por código.

Por ejemplo, si se quiere imitar el comportamiento del giroscopio, es necesario mover el dispositivo, quizás haya formas de emularlo por código, pero al final no va a ser una prueba que se acerque al uso real de un humano.

Otro ejemplo es escanear un código QR, que para automatizarlo se necesita que el celular esté bien soportado y que la distancia de lectura esté dentro de un rango específico.

Por lo tanto, siempre es importante que puedas analizar los beneficios y riesgos en una prueba automatizada.

**Limitaciones comunes de las pruebas automatizadas**

Al momento de pensar en la automatización de pruebas, ten presente las siguientes limitaciones antes de diseñar las pruebas:

* Solo se puede probar funcionalidades cuyo resultado sea interpretable por la máquina:
  + Que los resultados sean interpretables por la máquina quiere decir que el criterio de aceptación elegido debe ser medible automáticamente.

Por ejemplo, no se puede medir si el usuario está feliz al usar la aplicación. Lo que sí se puede cuantificar es el color, el número de clics, o si le salió una alerta, entre otros.

* Las pruebas automatizadas no reemplazan las pruebas exploratorias:
  + Las pruebas exploratorias son las acciones manuales que realiza una persona. La curiosidad, impredecibilidad e imaginación humana no se pueden automatizar.
* La Experiencia de Usuario (UX) no es medible:
  + No se puede medir ni automatizar la experiencia de usuario. Solo es posible recopilar datos para realizar análisis futuros.

**4 -. Pruebas funcionales y no funcionales**

* **Pruebas funcionales**: Permiten analizar los requisitos comerciales, es decir, lo que el cliente observara en la aplicación.
* **Pruebas no funcionales**: Permiten probar las cosas que son necesarias para que el cliente tenga una experiencia agradable, por ejemplo: el rendimiento, la seguridad y el almacenamiento de datos, entre otros.

**Tipos de Pruebas automatizadas**

* **Pruebas unitarias**: Las pruebas unitarias son las que se practican sobre bloques unitarios de software que pueden ser automatizados.

Este tipo de pruebas son las más fáciles de desarrollar y las más económicas.

Usualmente son realizadas por el equipo de desarrollo.

* **Pruebas de integración**: Las pruebas de integración son las que se hacen cubriendo multiples funcionalidades relacionadas, pudiendo automatizarlas en un grupo.
* **Pruebas de humo**: Las pruebas de humo (smoke tests) son pruebas que se ejecutan después de una fase de compilación para evitar los fallos o que se haya “incendiado” la aplicación (de ahí su nombre).
* **Pruebas de regresión**: Las pruebas de regresión consisten en asegurar que nuevos bloques de software no afecten a los antiguos, es decir, que no se añadan errores (bugs) en la aplicación o algo que estuviera funcionando deje de funcionar.
* **Pruebas de APIs**: Las pruebas de APIs consisten en asegurar de que los endpoints funcionen correctamente.

Las API’s o Application Programming Interfaces es la forma de conectar tu frontend o lo que el usuario observa con el backend (el servidor).

Podrás aprender más de este tema en Curso de Consumo de API REST con JavaScript

* **Pruebas de seguridad**: Las pruebas de seguridad o pentesting consiste en buscar las vulnerabilidades que puede tener un software o sistema.
* **Pruebas de rendimiento**: Las pruebas de rendimiento son pruebas no funcionales que consisten en evaluar la estabilidad y la capacidad de respuesta del software.

También se clasifican como pruebas de rendimiento, las pruebas de estrés, las cuales fuerzan la aplicación con condiciones de alto consumo, para analizar cómo se comporta en situaciones extremas.

* **Pruebas de aceptación**: Las pruebas de aceptación intentan determinar un criterio evaluable del cliente sobre la aplicación, es decir, cómo responderán al producto final.

Estas pruebas deben superarse con éxito antes del despliegue.

* **Pruebas de Interfaz Gráfica de Usuario (UI)**: Las pruebas de interfaces de usuario (UI) consisten en evaluar que el producto actúe correctamente.

Aquí se desarrollan las pruebas end-to-end, que consisten en analizar un flujo de principio a fin.

5 -. **Tipos de Frameworks automatizados**

**Capture/Playback**: El framework Capture/Playback consiste en grabar con una herramienta lo que el usuario ejecuta y reproducirlo como casos de prueba (scripts) grabados.

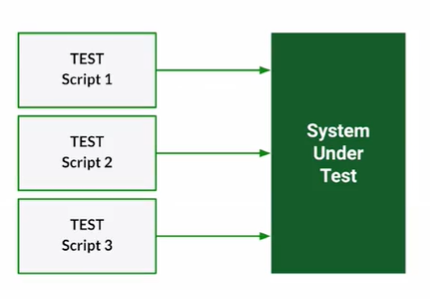
Las ventajas de capture/playback son:

* El enfoque captura/reproducción se puede utilizar para el System Under Test (sistema bajo prueba, SUT) en el nivel de una interfaz de usuario y/o API.
* Es fácil de configurar y usar para alguien sin experiencia previa.

Las desventajas de capture/playback son:

* Es difícil de mantener, porque es necesario grabar la aplicación cada vez que esta cambie.
* La implementación de los casos de prueba (scripts) solo puede comenzar hasta que el sistema esté disponible.

**Linear Scripting**: El framework Linear Scripting se asemeja a Capture/Playback, pero se diferencia en que el linear scripting permite observar y manipular el código grabado, mientras que en el Caputure/playback es inaccesible. Algunos autores los agrupan en el mismo tipo.



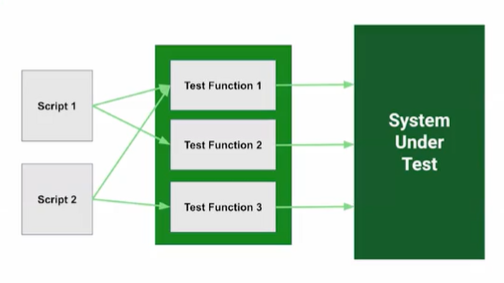
Las ventajas de linear scripting son:

* Se puede utilizar parar probar el sistema a nivel de interfaz de usuario y/o API.
* Fácil de configurar.

Las desventajas de linear scripting son:

* El costo de mantenimiento es lineal, no se pueden reutilizar los scripts, por lo que en productos grandes no es recomendable.
* La cantidad de esfuerzo es lineal, para cada caso de prueba necesitas un archivo de código.
* Es difícil de mantener, porque es necesario grabar la aplicación cada vez que cambie.

Structured Scripting: El framework Structured Scripting permite reutilizar el código de los casos de prueba.



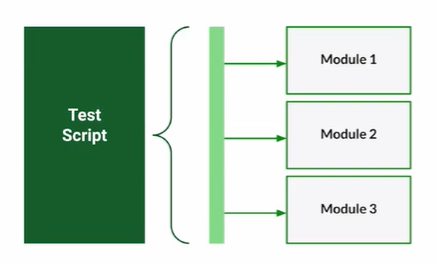
Las ventajas de *structured scripting* son:

* Existe una reducción del mantenimiento, por la reutilización de código.
* Reutilización de secuencias de comandos.
* Reducción de costos de construcción y manteniento.

Las desventajas de *structured scripting* son:

* Requiere un mayor esfuerzo inicial, tratar de dividir los *scripts* para que sean reutilizables.
* El equipo o encargado debe poseer habilidades de programación.
* Se necesita una buena administración.

Module Based: El framework Module Based se basa en módulos relacionados con los principios de la programación orientada a objetos (POO). Los módulos están separados por una capa de abstracción, de tal forma que los cambios puedan ser realizados en las secciones de la aplicación.



Las ventajas del *Module Based* son:

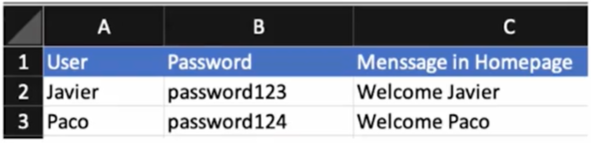
* Escalabilidad, podrás heredar atributos o propiedades en el código siguiendo la programación orientada a objetos.
* Flexibilidad y facilidad de mantenimiento.
* Modularización.

Las desventajas de *Module Based* son:

* No se tiene flexibilidad en los datos, la información debe estar escrita en el código.

**Data-driven**: El framework Data-Driven se basa en una estructura semejante a la de Module Based, la diferencia es que maneja entradas de datos al ser ingresados por el usuario, sin la necesidad de tener la información en el código.

Por ejemplo, si se requiere probar un inicio de sesión, no es necesario ir al código y cambiar las credenciales manualmente, sino extraer la información de un archivo de Excel, bases de datos, API, entre otros.



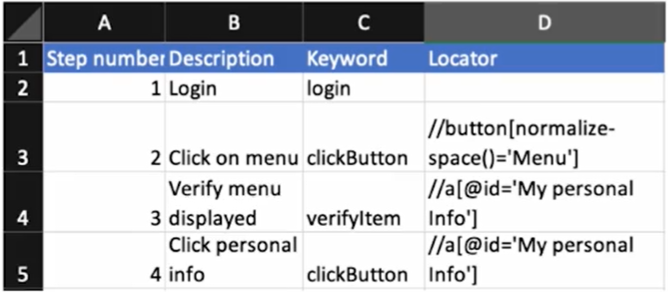
Las ventajas de *Data driven* son:

* Comparado con los *frameworks* anteriores, el costo se reduce considerablemente, ya que la información es más fácil de manejar.
* Aumenta la cobertura, porque será más fácil cubrir diferentes escenarios y probar la aplicación con diferentes valores.
* Flexibilidad de ejecución.

Las desventajas de *Data driven* son:

* Esfuerzo adicional para establecer, configurar y mantener las fuentes de datos.
* Dominio de algún lenguaje de programación.

**Keyword-driven**: El framework Keyword-Driven consiste en realizar acciones en el código a través de una palabra reservada (keyword). Lo que hará será activar una determinada acción según el procedimiento (step number), por ejemplo en la siguiente imagen observarás que primero se ejecutará un inicio de sesión (login), después un clic en un botón (clickButton) y así sucesivamente.



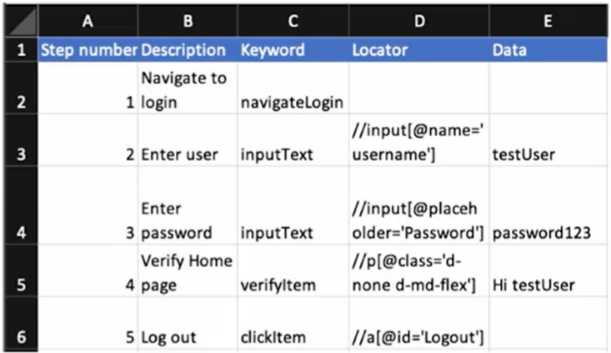
Las ventajas de *Keyword-driven* son:

* Flexibilidad en la creación de pruebas, porque puedes utilizar cualquier combinación de pasos.
* Reutilización de código y, por lo tanto, el costo de mantenimiento es menor.
* No se requiere conocimiento de las secuencias de comando, solo se necesita conocer qué hace cada palabra clave *(keyword)*.

Las desventajas de *Keyword-driven* son:

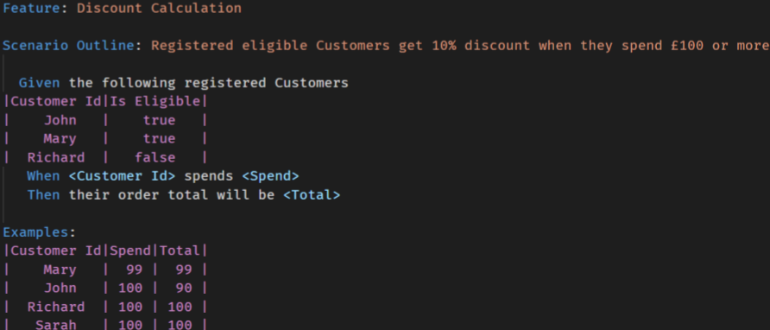
* Incremento de dificultad a medida que se introducen nuevas palabras clave.
* Complejidad de aprendizaje.

**Hybrid (Data-driven + Keyword-driven)**: El framework Hybrid consiste en una combinación de Data-driven y Keyword-driven, en el cual se tiene la información que desea utilizar en una acción determinada por la palabra clave (keyword) en un determinado paso (step number).



Behavior Driven Development (BDD): El framework Behavior Driven Development (BDD) es un marco de desarrollo impulsado por el desarrollo que contiene un lenguaje sencillo de leer para cualquiera. Normalmente, se utiliza un lenguaje natural Gherkin que consta de tres partes:

* Given: nos da las precondiciones de la prueba.
* When: nos da el detonante de la acción.
* Then: permite validar con un criterio de aceptación.



Las ventajas de *BDD* son:

* Existe una mayor compatibilidad entre historias de usuario y casos de prueba *(test cases)*.
* Claridad en los casos de prueba.
* Hay una reutilización de código, porque cada sentencia tiene un componente reutilizable.
* Es Data-Driven, porque tiene una funcionalidad llamada escenarios *outline* que permiten guardar información y utilizarla.

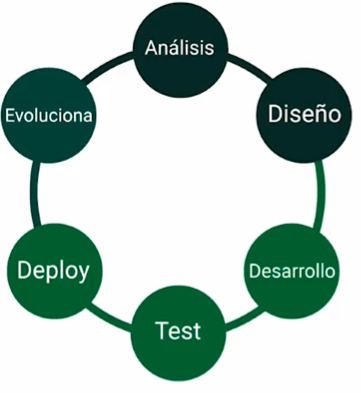
Las desventajas de *BDD* son:

* Mayor dedicación de tiempo.
* Mayor tiempo de planificación en la estructura del código y sus sentencias.
* Conocimiento de *Gherkin* o similar.

**6 -. Proceso de automatización de pruebas**

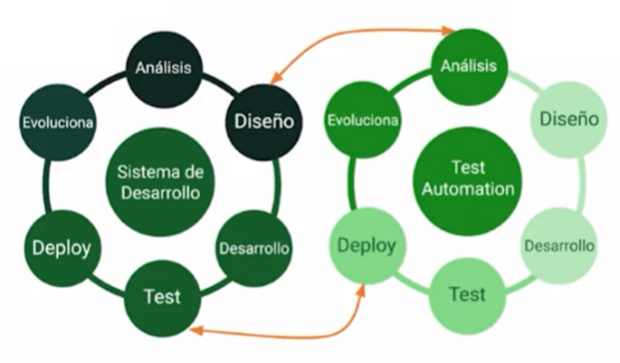
El proceso de automatización de pruebas consiste en integrar tus conocimientos de automatización, ventajas, desventajas, frameworks con tu flujo de trabajo o el de la empresa.

**Ciclo de desarrollo de una aplicación o de una solución automatizada**



**¿Cómo se integra con otras metodologías?**

El proceso de automatización de pruebas se integra con otras metodologías, como Scrum, de la siguiente manera:

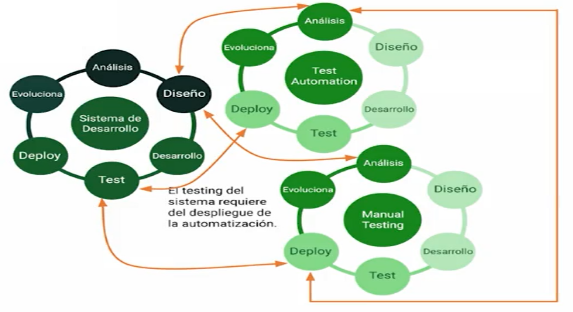


-El análisis de la automatización se basa en el diseño del sistema de desarrollo

-El test requiere del despliegue de la automatización, ya que no es posible probar el sistema si la solución de la prueba no está lista para utilizarla.

**¿Cómo se integra con un equipo de pruebas manuales?**

El proceso de automatización de prueba se integra con otras metodologías y con pruebas manuales de la siguiente manera:

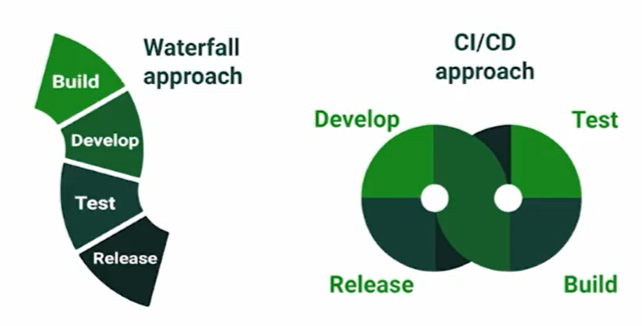


* El diseño del sistema también va a mejorar el análisis de las pruebas manuales.
* El despliegue de las pruebas manuales mejorará el análisis de la automatización de pruebas.

**7 -. Automatización e pruebas de CI/CD**

La industria del software evoluciona rápidamente, por lo tanto, se debe construir y entregar valor constantemente. La integración continua y el despliegue continuo (CI/CD) consiste en **distribuir las aplicaciones lo más rápido posible** con la ayuda de la automatización de pruebas.

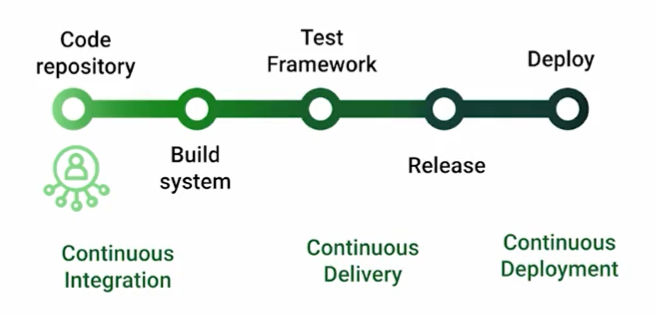
En la siguiente imagen observarás la forma convencional *(Waterfall approach)* frente a la integración y despliegue continuo (CI/CD). La diferencia principal es la interconectividad del desarrollo del CI/CD.



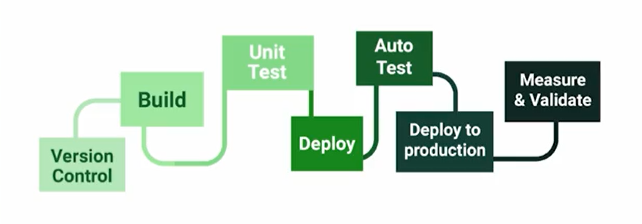
**Ejemplos de soluciones automatizadas**

El primer ejemplo consiste en un repositorio de código (GitHub, GitLab, entre otros), en el cual se construyen las soluciones *(Build system)*, después se implementan las pruebas *(Test framework)*, y finalmente se libera y despliega.

Esto ocurre cuando realizas un *commit* a una rama principal, entonces se ejecuta el *build*, después se realizan las pruebas. Esto permite integrar, liberar y desplegar de manera constante.



Pero esto puede cambiar dependiendo de las necesidades de la aplicación o de la empresa, por lo que puedes añadir más pasos, otras pruebas, validaciones, pero manteniendo el despliegue continuo.



**Beneficios de la automatización de pruebas en CI/CD**

Las ventajas que conlleva una integración y despliegue continuo son:

* Mayor agilidad para un desarrollo más rápido.
* Disminución de costos por la automatización de pruebas manuales y por la rapidez de entrega de valor de un producto.
* Mayor seguridad en el despliegue a producción, evitando introducir *bugs* al momento de entregar el producto.
* Aumento en la productividad y disminución de tiempos, permitiendo a los desarrolladores dedicar más tiempo a la solución de otros errores.

Finalmente, **no existen desventajas** en el uso de pruebas automatizadas en el flujo de integración y despliegue continuo.

8 -. Herramientas para la automatización de software

Unit testing

* Jest (javascript)
* Mocha (javascript)
* React/Vue Testing Library (frontend)
* Enzyme (javascript)

API Testing

* Rest assured
* Postman
* Insomnia

Web Browser testing

* Selenium
* Puppeteer
* Playwright
* Testcafe
* Protactor

Mobile testing

* Appium
* Detox
* Calabash

Performance

* Jmeter
* Gatling

Con Python tenemos:

Pruebas Unitarias:

Unittest

Pytest

Web browser Testing:

Selenium webdriver.

Api testing:

Locust

requests