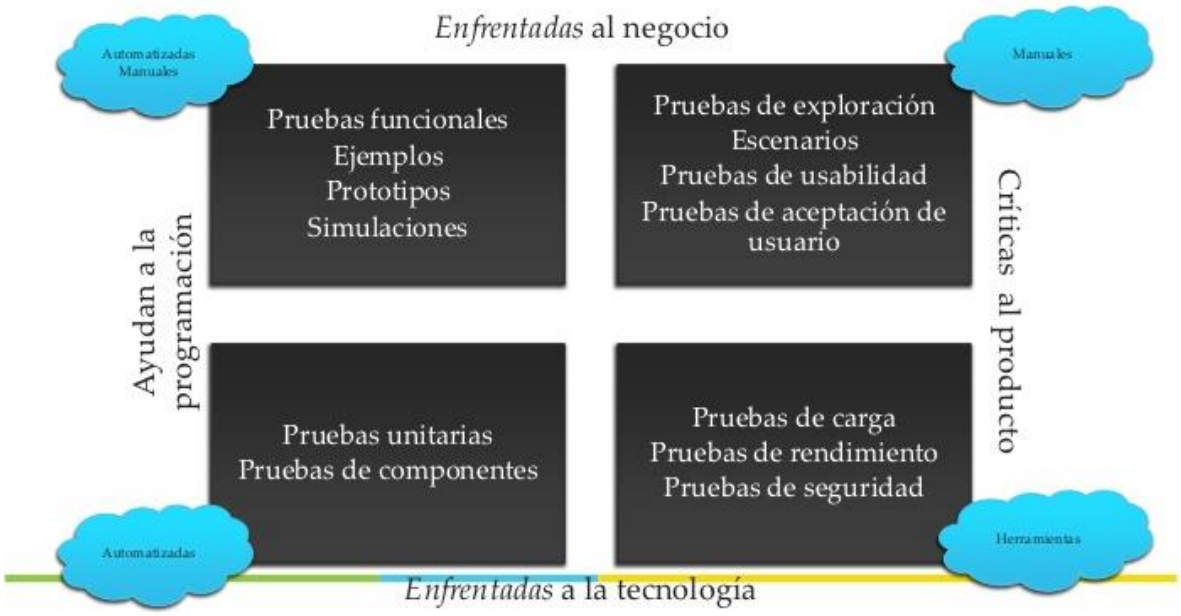
Pruebas automáticas

Un método de ejecutar una prueba sin intervención humana, que de lo contrario lo requeriría.

CUADRANTE DE AGILE TESTING



Cuadrante 1

Encontramos las pruebas unitarias y de componente. Estas pruebas se deberían intentar automatizar en el mayor porcentaje posible, para conseguir una situación de control sobre el trabajo ya desarrollado que aporte confianza al equipo para continuar trabajando "hacia delante".

Cuadrante 2

En este cuadrante encontramos las pruebas que, aun no siendo unitarias, prueban el sistema a un nivel más técnico que las pruebas que pudiera realizar un usuario final.

Cuadrante 3

Aquí se reúnen las pruebas que se han de realizar de forma manual, y en las que enfocándonos en el negocio, probaremos la aplicación con un espíritu crítico. A diferencia de los anteriores cuadrantes, las pruebas de este tendrán el objetivo de descubrir información sobre la aplicación y su comportamiento y no validar únicamente que se cubre las expectativas del programador.

Cuadrante 4

En este cuadrante se engloban las pruebas de rendimiento y todas las pruebas que puedan ser necesarias en el proyecto (usabilidad, seguridad, estabilidad, etc.).

¿POR QUÉ AUTOMATIZAR?

La principal razón es el tiempo y con las pruebas automatizadas se puede reducir el tiempo de las pruebas. Además, al automatizar las actividades comunes que no requieren de inteligencia humana, los testers reales pueden dedicar mayor tiempo a pruebas más críticas y caminos más elaborados dejando los caminos básicos a las pruebas automatizadas.

ATRIBUTOS DE AUTOMATIZACIÓN DE PRUEBAS

- Conciso: la prueba debe ser lo más sencillo posible y simple.
- Comprobación computacional: la prueba debe reportar sus resultados de forma que no se necesite interpretación humana.
- Repetible: la prueba se puede ejecutar varias veces sin la intervención humana.
- Robusto: test produce el mismo resultado ahora y para siempre. Las pruebas no se ven afectadas por cambios en el entorno externo.
- Necesario: todo en cada prueba contribuye a la especificación deseada de comportamiento.
- Despejado: cada afirmación es fácil de entender.
- Eficiente: las pruebas realizadas en un período razonable de tiempo.
- Específico: cada punto de falla de prueba a una parte específica de la funcionalidad rota (por ejemplo, cada caso de prueba comprueba un posible punto de fallo).
- Independiente: cada prueba se puede ejecutar por sí mismo o en una suite con un conjunto arbitrario de otras pruebas en cualquier orden.
- Sustentable: las pruebas deben ser fáciles de modificar y ampliar.
- Trazable: las pruebas deben ser conforme a los requisitos, los requisitos deben ser trazable a las pruebas.

¿QUÉ CASOS DE PRUEBA DEBERÍA AUTOMATIZAR?

- Casos de prueba que se deban correr en cada nueva versión de la aplicación (Sanity Testing).
- Casos de Prueba que utilicen distintos datos de prueba para las mismas acciones. (Data Driven Testing).
- Funcionalidades críticas de la aplicación (Smoke Testing).
- Funcionalidades que no cambiaran en un periodo de tiempo relativamente corto.
- Escenarios de pruebas de performance.

¿CUÁLES SON LAS DESVENTAJAS DE LA AUTOMATIZACIÓN?

- Las herramientas de testing automatizado no pueden medir la usabilidad de una aplicación.
- Se requieren conocimientos en programación para poder adaptar los scripts automatizados a los requerimientos.
- El mantenimiento de los scripts puede ser muy costoso.

¿QUÉ CANTIDAD DE CICLOS DE PRUEBA DEBE HABER?

Las pruebas automatizadas tienen más valor cada vez que se corren, por este motivo aquellos proyectos que involucren varios ciclos de prueba son los que sacan más valor de este tipo de pruebas.

¿QUÉ TECNOLOGÍA SE DEBE UTILIZAR?

Dependiendo de la tecnología que se utilice para desarrollar existen herramientas para automatizar de menor o mayor complejidad y de menor o mayor precio.

Las herramientas de testing automatizado pueden servir no solo para automatizar casos de prueba, sino que también para generar datos para los casos de prueba. Por este motivo conocer de la herramienta de automatización ayudará a también en las pruebas manuales.

¿CÓMO FORMAR A LAS PERSONAS QUE VAN A AUTOMATIZAR?

El proceso que siga cada una de las personas que vaya a automatizar puede variar según su grado de conocimiento en testing automatizado y en herramientas de automatización. Es aconsejable que de manera paralela puedan comenzar con la

capacitación en la herramienta que se vaya a utilizar, así como también leyendo material acerca de metodología y experiencias de pruebas automatizadas en general.

¿CÓMO SELECCIONAR LOS CASOS DE PRUEBA A AUTOMATIZAR?

El error más común es intentar automatizar todo, no intente automatizar todos los casos de prueba.

Teniendo una visión del negocio y también de la estructura interna del sistema (para lo cual se necesita involucrar a los desarrolladores) se debe decidir cuales casos de prueba que se automatizarán.

Algunas de los factores a tener en cuenta son:

- ¿Cuáles casos de prueba se necesitarán correr más veces durante el proyecto? Típicamente aquellos casos de prueba asociados al núcleo de la aplicación se pueden construir al principio del proyecto y luego sirven (adaptándolos cuando sea necesario) para el resto de la vida del sistema.

- ¿Cuáles casos de prueba conllevan mucha dedicación humana y son repetitivos? Hay veces que un caso de prueba requiere de un trabajo tedioso para configurar el ambiente de prueba y luego ejecutar el caso de prueba. Estos casos de prueba es bueno automatizarlos para liberar el recurso humano y que pueda dedicarse a otros casos más desafiantes.

- ¿Qué funcionalidades son críticas para el cliente/usuario? Aquellas funcionalidades que tienen que andar si o si en cada entrega generalmente es interesante incluirlas en los casos automatizados.

- ¿Qué costo tienen asociado la automatización del caso de prueba? Hay casos de prueba que son muy difíciles de automatizar, ya sea porque hay que realizar validaciones visuales complejas o por otras razones. Por este motivo si el costo de su automatización es grande hay veces que es mejor dejarlo para ser ejecutado manualmente.

Teniendo en cuenta estas características se puede ponderar cada caso de prueba en base a cada una de ellas para luego hacer un ranking o priorizar los casos de prueba a automatizar.

TABLA DE COMPARACIÓN DE TESTING AUTOMATIZADO VS TESTING MANUAL

Testing automatizado	Testing manual
Ventajas	**Ventajas**
Si se tienen que correr un juego de pruebas repetidamente, la automatización será de muchísima ayuda.	Si el caso de prueba sólo se ejecuta dos veces ante un cambio en la codificación, probablemente debería ser una prueba manual antes que automatizada ya que el costo es mayor al beneficio.

Permite correr las pruebas automatizadas sobre un código que cambia frecuentemente reduciendo el tiempo insumido en las pruebas de regresión.	Permite al tester ampliar más las pruebas durante la ejecución del caso de prueba. Encontrando más bugs ya que el tester puede inventar combinaciones impensadas mientras navega la aplicación y ante diversas situaciones.
Permite realizar matrices de pruebas combinando diferentes lenguajes con diferentes SO.	
Permite realizar pruebas en paralelo en una o varias máquinas.	
Desventajas	**Desventajas**
El esfuerzo inicial es mayor. La creación del script automatizado puede ser más costoso que la creación. Del caso de prueba para ejecución manual.	Las pruebas manuales pueden consumir demasiado tiempo.
No se pueden (o puede ser muy costoso) automatizar referencias visuales como colores o ubicación en pantalla de objetos.	Cada vez que hay un cambio en la aplicación el tester debe correr las pruebas de regresión. Ante reiterados cambios, el tester puede llegar a correr demasiadas veces las pruebas de regresión, reduciendo su capacidad de encontrar errores.

TIPOS DE HERRAMIENTAS DE TESTING

La selección de herramientas es un trabajo que cualquier líder de un proyecto de testing deberá utilizar. Por ello se debe familiarizar las herramientas del mercado y trabajar en como asociar sus características con las actividades estándar de testing.

A continuación, una descripción de varias herramientas del mercado con algunas características de ella agrupadas en las utilizadas en la gestión del testing, UI Testing y automatización.

Gestión del testing

Systir

- Testeo de sistemas especificando un lenguaje orientado al dominio.
- El tester utiliza este lenguaje (creado con Ruby).
- Esta especificación es ejecutable.
- Separación entre la especificación del test y la implementación.
- Browse.

TestLog

- Básica y fácil de usar.
- Repositorio XML (no BD SQL).
- Free.

Salomé – TMF

- Basado en el estándar para testing ISO9646:
  - Test Family
  - Test Suite
  - Test
  - Dataset
  - Execution
  - Test Campaign
  - Environment
- El test (case) puede ser manual o automático (JUnit, Abbot, CLIF, otro).
- El test automático es un script que será ejecutado por la herramienta y el resultado registrado en la campaña de prueba correspondiente.
- Free.

Test Track (Seapine)

- Workbook: dashboard de tareas, test cases, defects y runs asignados. Se pueden agregar tlabs.
- Folders: vista que permite organizar la información usando drag&drop.
- Custom fields.
- Linking defects.
- Automation rules.
- RSS.
- Soporta automatización mediante Automation Rules y workflow.

SQADesign

- Método Bottom-Up.
- Buena captura de UI (opcional) – no macros.
- Casos de uso (scenarios).
- Test Step (opcionales - UI o no).

Conexión con JIRA

- A nivel proyecto.
- Asociación de Test Cases, Scenarios, y Test Plans con JIRA issues.
- Carga de cualquier tipo de issue.
- Búsqueda de issues.
- Hecha en .Net y SQL server.
- Fácil de usar.

Xstudio

- Basada en modelo ISO.
- Interface basada en árboles.
- Administra requerimientos.
- Permite distribuir la ejecución de test suites en varias PC.
- Localizado para varios idiomas.
- No hay conexión con JIRA.
- Soporta parameters (y checks).
- Trazabilidad: un Test se asocia a una specification (y esta a los requerimientos).
- Tiene una opcion para generar requirement book (configurables vía XSLT).
- Importación: CSV, XML. Exportación: XML.
- Dependencias entre tests (execute only child tests if the parents are all successful).

- Lastima la implementación (deja ventana MS-DOS abierta).
- Free.

TestLink

- Última versión instalada en producción.
- Integración con JIRA (defects).
- ¿Automatización? (sólo permite leer los resultados de script externo).

UI Testing

Squish

- Testing Java GUI applications based on SWT, RCP/Eclipse, Swing and AWT.
- Los test pueden ser manuales o capturados (recording).

Borland The Open Alm Company

- Cada test case es independiente del resto.
- Ambiente de datos para cada uno.
- Se registra los test cases a partir de la UI y los valores esperados.

Abbot es un framework de testing programático tipo JUnit. De esta manera se puede realizar testing de regresión del look de la UI.

Salomé-TMF, que también administra casos de prueba JUnit y manuales, usa Abbot.

Automatización

Fitnesse

- Framework de test integrado a un wiki.
- Prueba de aceptación en lugar de unidad.
- Usa FIT
    - Especificación casos de prueba en MS-Excel
    - Implementación en JUNIT
- Filosofía de y para métodos ágiles.

TestMentor

- Administración, y generación de la automatización de componentes Java.
- Orientado a ser un puente entre los testers y los developers, permite definir assets de testing con la herramientas.
- Los test que se crean manualmente o que se generan pueden representarse en forma de código Java o en forma visual como activos de test.
- Muy completa.
- Lástima que no se integra con Eclipse IDE.
- Integra JUnit.

iValidator

- Framework de testing para Java.
- Se codifica las unidades de test en Java.
- Describir escenarios de prueba.
- Ejemplos:
    - Adapter to connect to the SUT
    - Units to perform tests
    - XML-TestDescriptions for several scenarios
    - Test Runner
- Se integra con JUnit.
- Existe plugin para Eclipse.

JML / ModernJass

- Java Modeling Language: lenguaje alto nivel diseñar por contratos con Java.
- Se usa tanto para diseñar como para testing (se puede reemplazar ciertos Junits)
- Pre/Post Conditions. Invariantes.
- Compilador, analizador estático, Runtime Checker.
- Plugins para Eclipse.

- Modern Jass: una implementación de JML5 que usa Java 5 annotations.
- Interesante para definición de contratos de subsistemas, programación defensiva sin afectar performance, mejor aprovechamiento de JUnits, testing funcional, componentes, exploratorio (runtime).