***Información sobre tipos de testing***

***Actividad 1 recopilar información sobre***

***- Pruebas unitarias***

Las pruebas unitarias consisten en aislar una parte del código y comprobar que funciona a la perfección. Son pequeños tests que validan el comportamiento de un objeto y la lógica.

El unit testing suele realizarse durante la fase de desarrollo de aplicaciones de software o móviles. Normalmente las llevan a cabo los desarrolladores, aunque en la práctica, también pueden realizarlas los responsables de QA.

Hay una especie de mito respecto a las pruebas unitarias. Algunos desarrolladores están convencidos de que son una pérdida de tiempo y las evitan buscando ahorrar tiempo.

Nada más alejado de la realidad.

Con ellas se detectan antes errores que, sin las pruebas unitarias, no se podrían detectar hasta fases más avanzadas como las pruebas de sistema, de integración e incluso en la beta.

Realizar pruebas unitarias con regularidad supone, al final, un ahorro de tiempo y dinero.

Motivos para realizar un test unitario

* Las pruebas unitarias demuestran que la lógica del código está en buen estado y que funcionará en todos los casos.
* Aumentan la legibilidad del código y ayudan a los desarrolladores a entender el código base, lo que facilita hacer cambios más rápidamente.
* Los test unitarios bien realizados sirven como documentación del proyecto.
* Se realizan en pocos milisegundos, por lo que podrás realizar cientos de ellas en muy poco tiempo.
* Las *unit testing* permiten al desarrollador refactorizar el código más adelante y tener la garantía de que el módulo sigue funcionando correctamente. Para ello se escriben casos de prueba para todas las funciones y métodos, para que cada vez que un cambio provoque un error, sea posible identificarlo y repararlo rápidamente.
* La calidad final del código mejorará ya que, al estar realizando pruebas de manera continua, al finalizar el código será limpio y de calidad.
* Como las pruebas unitarias dividen el código en pequeños fragmentos, es posible probar distintas partes del proyecto sin tener que esperar a que otras estén completadas.

**Buenas prácticas para los tests unitarios**

Una vez te acostumbres a realizar este tipo de pruebas irás descubriendo todas sus ventajas. Sin embargo, si todavía no tienes experiencia, vamos a ver algunos ejemplos de buenas prácticas en las pruebas unitarias de software.

* Las pruebas unitarias deberían ser independientes. Si se produce cualquier tipo de mejora o cambio en los requerimientos, las pruebas unitarias no deberían verse afectados.
* Prueba sólo un código a la vez.
* Sigue un esquema claro. Puede parecer algo secundario, pero no lo es. Sé también consistente a la hora de nombrar tus *unit tests*.
* Cualquier cambio necesita pasar el test. En el caso de producirse un cambio en el código de cualquier módulo, asegúrate de que hay una prueba unitaria que se corresponda con ese módulo y que este pasa las pruebas antes de cambiar la implementación.
* Corrige los bugs identificados durante las pruebas antes de continuar. Asegúrate de realizar esta corrección antes de proseguir con la siguiente fase del ciclo de vida del desarrollo de software.
* Acostúmbrate a realizar pruebas regularmente mientras programas. Cuanto más código escribas sin testar, más caminos tendrás que revisar para encontrar errores.

No puedes esperar que las pruebas unitarias descubran todos los errores de un software, pero sí que ahorran mucho tiempo al facilitar localizar los errores de una manera más sencilla.

***- Pruebas de integración***

Las pruebas de integración implican probar diferentes módulos de una aplicación de software como grupo. Una aplicación de software se compone de diferentes submódulos que trabajan juntos para diferentes funcionalidades. El propósito de las pruebas de integración es validar la integración de diferentes módulos juntos e identificar los errores y problemas relacionados con ellos.

* Se asegura de que todos los módulos de aplicación estén bien integrados y funcionen juntos según lo esperado.
* Detecta problemas y conflictos interconectados para resolverlos antes de crear un gran problema.
* Valida la funcionalidad, fiabilidad y estabilidad entre diferentes módulos.
* Detecta excepciones ignoradas para mejorar la calidad del código.
* Admite la canalización de CI/CD.

Debido a que en las pruebas unitarias es necesario crear módulos auxiliares que simulen las acciones de los componentes invocados por el que se está probando y a que se han de crear componentes «conductores» para establecer las precondiciones necesarias, llamar al componente objeto de la prueba y examinar los resultados de la prueba, a menudo se combinan los tipos de prueba unitarias y de integración.

Los tipos fundamentales de integración son los siguientes:

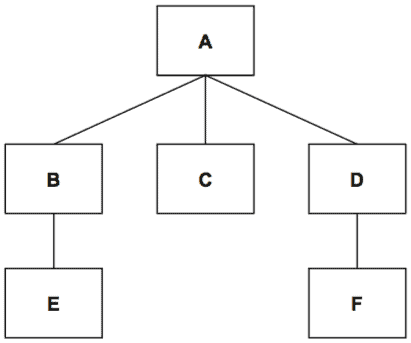
*Integración incremental*

Se combina el siguiente componente que se debe probar con el conjunto de componentes que ya están probados y se va incrementando progresivamente el número de componentes a probar.

Con el tipo de prueba incremental lo más probable es que los problemas que surjan al incorporar un nuevo componente o un grupo de componentes previamente probado, sean debidos a este último o a las interfaces entre éste y los otros componentes.

Se distinguen las siguientes estrategias de integración:

* **De arriba abajo (*top-down*)**. El primer componente que se desarrolla y prueba es el primero de la jerarquía (A). Los componentes de nivel más bajo se sustituyen por componentes auxiliares para simular a los componentes invocados. En este caso no son necesarios componentes conductores. Una de las ventajas de aplicar esta estrategia es que las interfaces entre los distintos componentes se prueban en una fase temprana y con frecuencia.
* **De abajo arriba (*bottom-up*)**. En este caso se crean primero los componentes de más bajo nivel (E, F) y se crean componentes conductores para simular a los componentes que los llaman. A continuación se desarrollan los componentes de más alto nivel (B, C, D) y se prueban. Por último dichos componentes se combinan con el que los llama (A). Los componentes auxiliares son necesarios en raras ocasiones.
* Este tipo de enfoque permite un desarrollo más en paralelo que el enfoque de arriba abajo, pero presenta mayores dificultades a la hora de planificar y de gestionar.
* **Estrategias combinadas**. A menudo es útil aplicar las estrategias anteriores conjuntamente. De este modo, se desarrollan partes del sistema con un enfoque «*top-down*«, mientras que los componentes más críticos en el nivel más bajo se desarrollan siguiendo un enfoque «*bottom-up*«. En este caso es necesaria una planificación cuidadosa y coordinada de modo que los componentes individuales se «encuentren» en el centro.



*Integración no incremental*

Se prueba cada componente por separado y posteriormente se integran todos de una vez realizando las pruebas pertinentes. Este tipo de integración se denomina también *Big-Bang* (gran explosión).

***- Pruebas de usabilidad***

La usabilidad indica la facilidad de uso de una herramienta. Abarca tanto la experiencia del usuario como la sencillez para lograr un objetivo por medio de un sistema o dispositivo, por lo que es aplicable tanto a software como a hardware.

Los 10 principios de la usabilidad

[**Jakob Nielsen**](https://www.nngroup.com/articles/ten-usability-heuristics/), experto en usabilidad, enunció los principios que te ayudarán a evaluar tu diseño web:

1. **Visibilidad del estado del sistema:**tu usuario sabe exactamente qué está pasando, en el momento preciso. *Ejemplo: al rellenar un formulario, el usuario sabe si el envío fue procesado y cuándo lo contactarás.*
2. **Relación entre el sistema y el mundo real:** la información está disponible en un orden lógico y en un lenguaje convencional. *Ejemplo: eliminas el lenguaje técnico en las páginas dedicadas a los consumidores finales.*
3. **Control y libertad del usuario:** tus usuarios pueden rehacer o deshacer sus pasos al navegar. *Ejemplo: pones un botón de «Ir atrás» en una página de producto*.
4. **Estándares y consistencia:**cada término o situación está utilizado de manera unívoca, y no hay formas distintas de nombrar o mostrar una misma cosa. *Ejemplo: haces que todos los botones de «Seguir» tengan la misma apariencia.*
5. **Prevención de errores:** hay varias formas de evitar los errores en la plataforma. *Ejemplo: añades un mensaje de «Confirmar envío de los datos» antes que el usuario pase a la etapa siguiente.*
6. **Reconocimiento en vez de recuerdo:** tus usuarios no deben memorizar continuamente lo que deben hacer, ya que tienen sus opciones visibles. *Ejemplo: instalas botones con iconos fácilmente reconocibles.*
7. **Flexibilidad y eficiencia en el uso:** tus distintos tipos de usuarios encuentran un uso adecuado a sus necesidades.*Ejemplo: tienes opciones visibles para los usuarios nuevos, a la vez que cuentas con atajos para los más experimentados.*
8. **Diseño minimalista y estético:**cada pieza de información es relevante. Lo superfluo es eliminado. *Ejemplo: eliminas todos los botones que son raramente utilizados, con el fin de que las opciones más importantes aparezcan de manera clara.*
9. **Asistencia a los usuarios para reconocer, diagnosticar y corregir los errores:** indicas los errores de manera comprensible, sin lenguaje técnico y con una solución rápida. *Ejemplo: haces que tus páginas rotas no muestren un simple código, sino que explicas al usuario qué significa y qué puede hacer.*
10. **Ayuda y documentación:**los usuarios disponen de una base de conocimiento enfocada en acciones. *Ejemplo: añades una lista de preguntas frecuentes.*

**Cómo hacer pruebas de usabilidad: 9 fases**

1. Decide qué parte de tu producto o sitio web quieres probar

¿Tienes alguna pregunta urgente sobre cómo interactúan los usuarios con ciertas partes de tu diseño, una interacción o flujo de trabajo en particular? ¿Te preguntas qué harán los usuarios primero cuando lleguen a la página o conozcan tu producto? Reúne tus ideas sobre las ventajas, desventajas y áreas de mejora de tu producto o sitio web para que puedas crear una hipótesis sólida para tu estudio.

2. Elige las tareas de tu estudio

Las tareas de tus participantes deben ser los objetivos más comunes de tus usuarios cuando interactúan con tu producto o sitio web, como hacer una compra, por ejemplo.

3. Establece un estándar para el éxito de cada tarea

Una vez que sepas qué probar y cómo probarlo, **asegúrate de establecer criterios claros para determinar el éxito de cada tarea.** Por ejemplo, en una de las pruebas de usabilidad para la herramienta [**Estrategia de Contenido de HubSpot**](https://www.hubspot.es/products/marketing/content-strategy), debíamos añadir una publicación de blog a un cluster e informar cada paso que dimos.

Si registras cuáles son los pasos que deben dar tus usuarios podrás establecer un umbral de éxito o fracaso para cada tarea, a la vez que determinar si la experiencia de usuario es lo suficientemente intuitiva o no.

4. Redacta un plan de estudio y un guion

Al comienzo de tu guion, debes incluir el propósito del estudio. Anuncia si vas a grabar las sesiones, da algunos antecedentes del producto o sitio web y haz preguntas para determinar cuánto saben los participantes acerca del objeto de estudio. Por último, los moderadores deben seguir el mismo guion en cada sesión de usuario, a fin de que el estudio sea coherente, imparcial y con un enfoque científico.

5. Delega las funciones

Durante el estudio de usabilidad, el moderador debe encaminar cuidadosamente a los participantes y seguir estrictamente el guion. Debe mantenerse neutral, no ceder ante la presión social y hacer que los participantes se sientan cómodos, mientras los impulsa a completar las tareas. Piensa en cuál miembro de tu equipo va mejor con este rol.

Tomar notas durante el estudio también es igual de importante. Si no hay datos registrados, no puedes extraer ninguna información que pruebe o refute tu hipótesis. El oyente más atento de tu equipo debe ser quien tome las notas.

6. Encuentra a tus participantes

La detección y reclutamiento de los participantes correctos es la parte más difícil de las pruebas de usabilidad. [**La mayoría de los expertos en usabilidad sugieren que solo debes evaluar a cinco participantes durante cada estudio**](https://www.invisionapp.com/inside-design/ux-usability-research-testing/), cuidando que se parezcan mucho a tu base de usuarios real. Con una muestra tan reducida será un reto replicarla.

Para reclutar a los participantes ideales para el estudio, crea el **[user/buyer persona](https://www.iebschool.com/blog/guia-crear-buyer-persona-inbound-marketing/)** más detallado y específico que puedas e incentiva a participar con una tarjeta de regalo u otra recompensa atractiva.

Reclutar colegas de otros departamentos que potencialmente usarían tu producto también es otra opción. Cuidado con dejar que uno de los miembros de tu equipo conozca a los participantes porque entonces esa relación personal puede crear un sesgo, ya que van a procurar ser amables entre sí: el investigador podría ayudar al usuario a completar una tarea o el usuario podría no querer criticar el diseño del producto del investigador.

7. Realiza el estudio

Durante el estudio, debes pedir a los participantes que completen una tarea a la vez, sin ayuda u orientación. Si el participante te pregunta cómo hacer algo, no digas nada, ya que es importante ver cuánto tiempo les lleva a los usuarios descubrir tu interfaz.

Pedirles a los participantes que «piensen en voz alta» es una táctica efectiva. De este modo sabrás lo que pasa por la cabeza de un usuario cuando interactúa con tu producto o sitio web.

Después de completar cada tarea, solicita comentarios acerca de si esperarían ver en el mercado tu producto tal cual es ahora; si habrían completado la tarea si no fuera una prueba; si recomendarían tu producto a un amigo y qué cambiarían. Estos datos cualitativos pueden ayudarte a identificar más ventajas y desventajas del diseño.

8. Analiza tus datos

Vas a recolectar un montón de datos cualitativos después del estudio. Analizarlos te ayudará a descubrir patrones de cualquier asunto, así como medir la gravedad de cada problema de usabilidad y proporcionar recomendaciones de diseño al equipo de ingeniería.

Cuando analices tus datos, asegúrate de prestar atención tanto al rendimiento de los usuarios como a [**sus sentimientos sobre el producto**](https://latinamericanpost.com/es/29676-en-diseno-de-productos-imaginar-los-sentimientos-del-usuario-es-la-clave). No es inusual que un participante logre su objetivo de manera rápida y exitosa, y aun así se exprese de forma negativa sobre la experiencia.

9. Informa tus hallazgos

Después de extraer información de los datos, informa las conclusiones principales. Determina los pasos a seguir para perfeccionar el diseño del producto o sitio web y las mejoras que esperas ver durante la próxima ronda de pruebas.

***Actividad 2 - Automatizaciones***

- Buscar técnicas, librerías, patrones para la automatización de dichas pruebas

En las [pruebas de software](https://es.wikipedia.org/wiki/Pruebas_de_software), la **automatización de pruebas** consiste en el uso de software especial (casi siempre separado del software que se prueba) para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados. La automatización de pruebas permite incluir pruebas repetitivas y necesarias dentro de un proceso formal de pruebas ya existente o bien adicionar pruebas cuya ejecución manual resultaría difícil.

Algunas [pruebas de software](https://es.wikipedia.org/wiki/Pruebas_de_software) tales como las [pruebas de regresión](https://es.wikipedia.org/wiki/Pruebas_de_regresi%C3%B3n) intensivas de bajo nivel pueden ser laboriosas y consumir mucho tiempo para su ejecución si se realizan manualmente. Adicionalmente, una aproximación manual puede no ser efectiva para encontrar ciertos tipos de defectos, mientras que las pruebas automatizadas ofrecen una alternativa que lo permite. Una vez que una prueba ha sido automatizada, esta puede ejecutarse repetitiva y rápidamente en particular con productos de software que tienen ciclos de mantenimiento largo, ya que incluso cambios relativamente menores en la vida de una aplicación pueden inducir fallos en funcionalidades que anteriormente operaban de manera correcta. Existen dos aproximaciones a las pruebas automatizadas:

* **Pruebas manejadas por el código**: Se prueban las [interfaces públicas](https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones) de las [clases](https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)), [módulos](https://es.wikipedia.org/wiki/M%C3%B3dulo_(inform%C3%A1tica)) o [bibliotecas](https://es.wikipedia.org/wiki/Biblioteca_(inform%C3%A1tica)) con una variedad amplia de argumentos de entrada y se valida que los resultados obtenidos sean los esperados.
* **Pruebas de**[**Interfaz de Usuario**](https://es.wikipedia.org/wiki/Interfaces_de_usuario): Un marco de pruebas genera un conjunto de eventos de la interfaz de usuario, tales como teclear, hacer click con el ratón e interactuar de otras formas con el software y se observan los cambios resultantes en la interfaz de usuario, validando que el comportamiento observable del programa sea el correcto.

La elección misma entre automatización y ejecución manual de pruebas, los componentes cuya prueba será automatizada, las herramientas de automatización y otros elementos son críticos en el éxito de las pruebas, y por lo regular deben provenir de una elección conjunta de los equipos de desarrollo, control de calidad y administración. Un ejemplo de mala elección para automatizar, sería escoger componentes cuyas características son inestables o su proceso de desarrollo implica cambios continuos.

**Pruebas manejadas por el código**

En el desarrollo contemporáneo de software existe una tendencia creciente a usar [Frameworks](https://es.wikipedia.org/wiki/Framework" \o "Framework) como los denominados [XUnit](https://es.wikipedia.org/wiki/XUnit" \o "XUnit) (por ejemplo [JUnit](https://es.wikipedia.org/wiki/JUnit" \o "JUnit) y [NUnit](https://es.wikipedia.org/wiki/NUnit" \o "NUnit)) que permiten la ejecución de [pruebas unitarias](https://es.wikipedia.org/wiki/Prueba_unitaria) para determinar cuándo varias secciones del código se comportan como es esperado en circunstancias específicas. Los [casos de prueba](https://es.wikipedia.org/wiki/Casos_de_Prueba) describen las pruebas que han de ejecutarse sobre el programa para verificar que este se ejecuta tal y como se espera. La automatización de pruebas es una característica clave del [desarrollo ágil de software](https://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software) en donde se le conoce como "[desarrollo guiado por pruebas](https://es.wikipedia.org/wiki/Desarrollo_guiado_por_pruebas)". En ellas, las pruebas unitarias se escriben **antes** que el código que genera la funcionalidad. Solo cuando el código pasa exitosamente las pruebas se considera completo. Cuando hay cambios, el programador descubre inmediatamente cualquier defecto que rompa los casos de prueba lo cual baja el costo de la reparación. Dos inconvenientes de este estilo de trabajo son:

1. Algunas veces se "desperdicia" la capacidad del programador escribiendo las pruebas unitarias. El entrecomillado se debe precisamente que asegurar la calidad del producto no es desperdicio alguno.
2. Normalmente se prueban los requerimientos básicos o el flujo normal del [caso de uso](https://es.wikipedia.org/wiki/Caso_de_uso) en vez de todos los flujos alternativos, dado que extender las pruebas más allá de la prueba base eleva el costo del producto. En algunas ocasiones los flujos alternativos son probados por un equipo de pruebas más o menos independiente del equipo de desarrollo.

**Pruebas de Interfaz de Usuario**

Muchas herramientas de automatización de pruebas proveen características para grabar y reproducir acciones del usuario para posteriormente ejecutarlas un número indefinido de veces, comparando resultados obtenidos con resultados esperados. La ventaja de esta aproximación a la automatización es que requiere de menos desarrollo de software, sin embargo, el confiar en estas características del software lo hace menos confiable en la medida que muchas veces dependen de la etiqueta o posición del elemento de interfaz, y, al cambiar, el caso de prueba debe ser adaptado al cambio o probablemente fallar. Una variante de estas pruebas es la prueba de sistemas basados en la web en las que la herramienta de prueba ejecuta acciones sobre el navegador e interpreta el HTML resultante. Una variación más es la automatización sin scripts, que no usa grabación y reproducción de acciones, sino que construye un modelo de la Aplicación Bajo Prueba ABP (AUT en sus siglas en inglés) que permite a la persona que verifica ("tester") crear pruebas simplemente editando parámetros y condiciones.

***Actividad 3 - Rendimiento***

- Buscar información sobre como realizar pruebas de rendimiento

La prueba de rendimiento es una técnica de [prueba de software](https://www.microfocus.com/products/performance-engineering/overview) no funcional que determina cómo la estabilidad, la velocidad, la escalabilidad y la capacidad de respuesta de una aplicación se mantiene bajo una determinada carga de trabajo. Es un paso clave para asegurar la calidad del software, pero desafortunadamente, a menudo se ve como una reflexión posterior, en aislamiento, y para comenzar una vez que se completan las pruebas funcionales, y en la mayoría de los casos, después de que el código está listo para ser distribuido.

Los objetivos de las [pruebas de rendimiento](https://www.microfocus.com/solutions/performance-testing) incluyen la evaluación de la salida de la aplicación, la velocidad de procesamiento, la velocidad de transferencia de datos, el uso del ancho de banda de la red, el máximo de usuarios concurrentes, la utilización de la memoria, la eficiencia de la carga de trabajo y los tiempos de respuesta de los comandos.

Razones para las pruebas de rendimiento

Las organizaciones realizan pruebas de rendimiento al menos por una de las siguientes razones:

* Para determinar si la aplicación satisface los requisitos de rendimiento (por ejemplo, el sistema debe manejar hasta 1.000 usuarios simultáneos).
* Para localizar los cuellos de botella informáticos dentro de una aplicación.
* Para establecer si los niveles de rendimiento reclamados por un proveeder de software son realmente verdaderos.
* Para comparar dos o más sistemas e identificar el que mejor funciona.
* Para medir la estabilidad en los casos de picos de tráfico.

¿Cómo realizar las pruebas de rendimiento?

Los pasos específicos de las pruebas de rendimiento variarán de una organización y aplicación a otra. Depende de los indicadores de rendimiento que la empresa considere más importantes. Sin embargo, los objetivos generales de las pruebas de rendimiento son en gran medida los mismos en general, por lo que hay un cierto flujo de trabajo que seguirán la mayoría de los planes de pruebas.

Identifique el entorno y las herramientas de prueba

Identifique el entorno de producción, el entorno de prueba y las herramientas de prueba a su disposición. Documente el hardware, el software, las especificaciones de infraestructura y las configuraciones tanto en los entornos de prueba como de producción para asegurar la coherencia. Algunas pruebas de rendimiento pueden tener lugar en el entorno de producción, pero deben existir estrictas medidas de seguridad que impidan que las pruebas afecten a las operaciones de producción.

Defina los criterios de rendimiento aceptables

Determine las limitaciones, objetivos y umbrales que demostrarán el éxito de la prueba. Los principales criterios se derivarán directamente de las especificaciones del proyecto, pero los encargados de las pruebas deben estar debidamente facultados para establecer un conjunto más amplio de pruebas y puntos de referencia.

Pruebas de planificación y diseño

Piense en la gran variedad de usos y luego cree escenarios de prueba que den cabida a todos los casos de uso posibles. Diseñe las pruebas en consecuencia y defina las métricas que deberían capturarse.

Prepare el entorno y las herramientas de prueba

Configure el entorno de prueba antes de ejecutar las pruebas de rendimiento. Reúna sus herramientas de prueba disponibles.

Ejecute las pruebas de rendimiento

Ejecute las pruebas. Capture y supervise los resultados.

Resuelva y vuelva a probar

Consolide y analice los resultados de las pruebas. Comparta los descubrimientos con el equipo del proyecto. Ajuste la aplicación resolviendo las deficiencias de rendimiento identificadas. Repita la prueba para confirmar que cada problema ha sido eliminado de manera definitiva.

Consejos para las pruebas de rendimiento

Cree un entorno de pruebas que refleje el ecosistema de producción lo más fielmente posible. Sin eso, los resultados de la prueba pueden no ser una representación exacta del rendimiento de la aplicación cuando se ponga en marcha.

* Separe el entorno de prueba de rendimiento del entorno de la UAT.
* Identifique las herramientas de prueba que mejor automatizan su plan de pruebas de rendimiento.
* Ejecute las pruebas varias veces para obtener una medida precisa del rendimiento de la aplicación. Si está realizando una prueba de carga, por ejemplo, realice la misma prueba varias veces para determinar si el resultado es consistente antes de marcar el rendimiento como aceptable o inaceptable
* No haga cambios en el entorno de la prueba entre las pruebas.

BIBLIOGRAFIA

<https://www.yeeply.com/blog/que-son-pruebas-unitarias/#:~:text=Las%20pruebas%20unitarias%20consisten%20en,aplicaciones%20de%20software%20o%20m%C3%B3viles>.

<https://blog.hubspot.es/marketing/pruebas-usabilidad#:~:text=La%20usabilidad%20indica%20la%20facilidad,a%20software%20como%20a%20hardware>.

Kolawa, Adam; Huizinga, Dorota (2007). *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press. p. 74. [ISBN](https://es.wikipedia.org/wiki/ISBN) [978-0-470-04212-0](https://es.wikipedia.org/wiki/Especial:FuentesDeLibros/978-0-470-04212-0).

<https://www.microfocus.com/es-es/what-is/performance-testing#:~:text=La%20prueba%20de%20rendimiento%20es,una%20determinada%20carga%20de%20trabajo>.