IT Bootcamp

Introducción a la Calidad del Software

DigitalHouse>

Índice

1. Introducción a la calidad del software
1. Calidad del código
2. Testing Automatizado vs Testing Manual

1 | Calidad del software

El glosario de la IEEE (Institute of Electrical and Electronics Engineers) para la ingeniería de software define la calidad del software como:

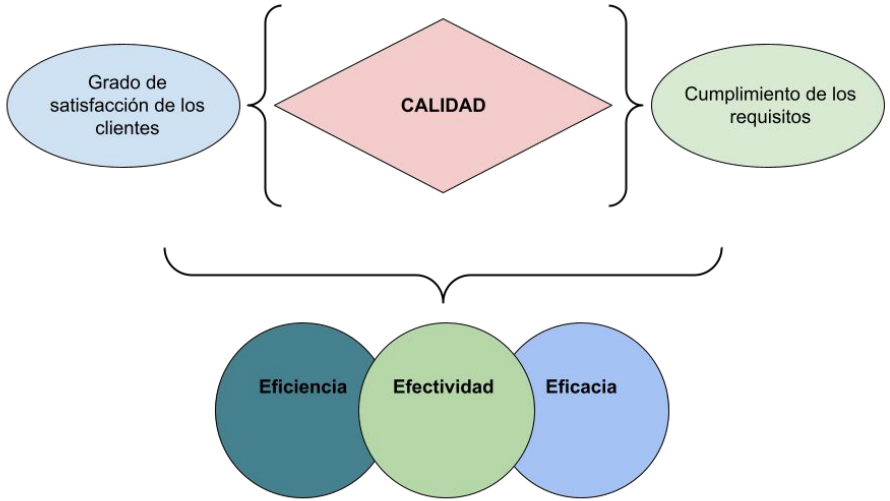
"

El **grado** con el cual un sistema, componente o proceso **cumple** con los requerimientos, las necesidades y/o expectativas del cliente o usuario.

"

Introducción a la calidad del software

La calidad del software es **medible** y varía de un sistema a otro, o de un programa a otro. Puede medirse después de elaborado el producto, pero esto puede resultar muy costoso si se detectan problemas derivados de imperfecciones en el diseño, por lo que es imprescindible tener en cuenta tanto la obtención de la calidad como su control durante todas las etapas del ciclo de vida del software.



Cumplir con los requisitos

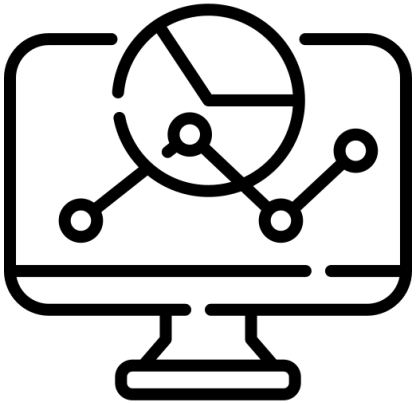
El grado de calidad de un producto software, debe ser medido en función al cumplimiento de los **requisitos funcionales** (describen lo que el sistema debería hacer, por eso se les dicen “EL QUÉ”) y **no funcionales** (establecen restricciones de cómo estos requisitos funcionales son implementados, por eso se les dicen “EL CÓMO”).

El cumplimiento de estos requisitos nos permitirá entender la calidad esperada por los usuarios desde enfoques diferentes.

¿Cómo obtener un software de calidad?

La obtención de un software con calidad implica la utilización de **metodologías o procedimientos estándares** para el análisis, diseño, programación y prueba del software que permitan uniformar la filosofía de trabajo, para lograr una mayor **confiabilidad, mantenibilidad y facilidad** de prueba, a la vez que eleven la productividad, tanto para el desarrollo como para el control de la calidad del software.

La adopción de una buena política contribuye en gran medida a lograr la calidad del software, pero no la asegura. **Para el aseguramiento de la calidad es necesario su control o evaluación**.



Entonces nos preguntamos... **¿Es posible encontrar un conjunto de propiedades en un software que nos den una indicación de su calidad?**

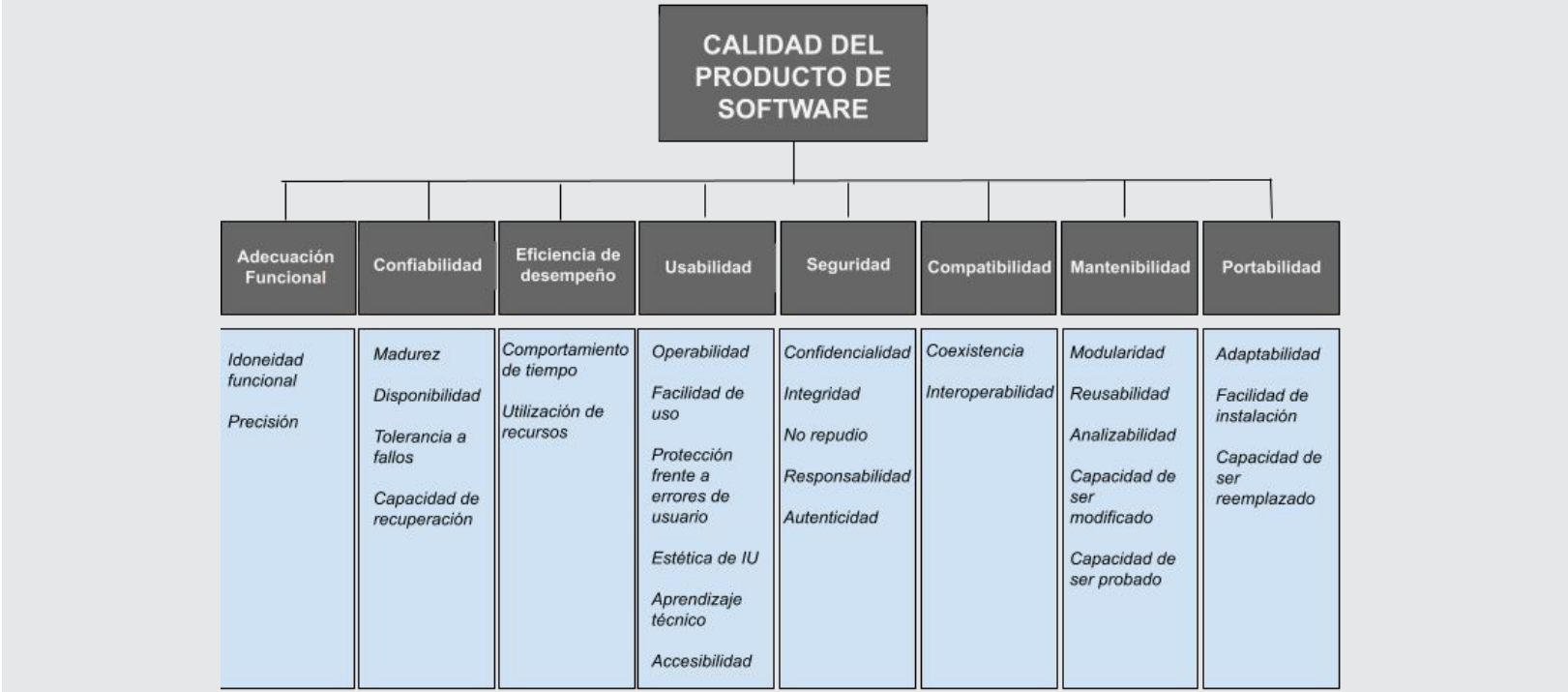
Para dar respuesta a esta pregunta aparecen los Modelos de Calidad. En los Modelos de Calidad, la calidad se define de forma jerárquica y tienen como objetivo resolver la complejidad mediante la descomposición.

Control de calidad del software

Para controlar la calidad del software es necesario, ante todo, definir los **parámetros, indicadores o criterios de medición**, ya que, como plantea Tom De Marco, *"no se puede controlar lo que no se puede medir"*.

Existen diversos estándares y modelos a considerar para evaluar la calidad de un software. A continuación se presenta el modelo de calidad definido por la ISO/IEC 25010, el cual se encuentra compuesto por ocho **características o dimensiones de calidad**.

Modelo de calidad del software ISO/IEC 25010



Control de calidad del software

Adecuación Funcional

Representa el grado en que un producto o sistema proporciona funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.

Eficiencia de desempeño

Esta característica representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.

Compatibilidad

Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.

Control de calidad del software

Usabilidad

Capacidad del software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.

Confiabilidad

Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.

Seguridad

Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no puedan leerlos o modificarlos.

Control de calidad del software

Mantenibilidad

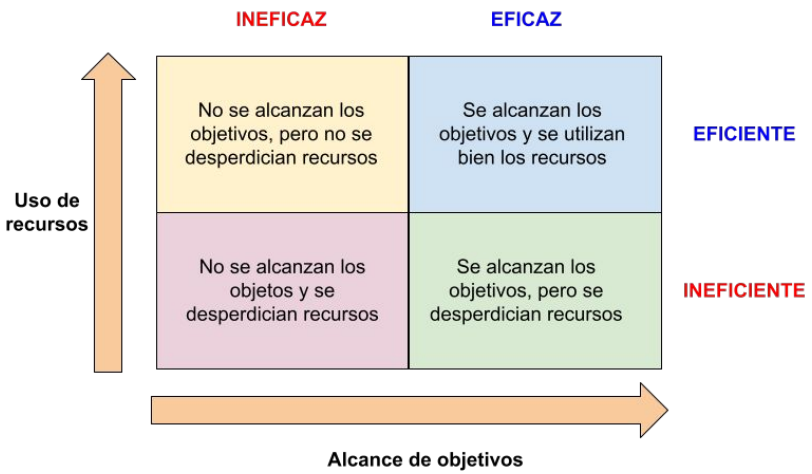
Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.

Portabilidad

Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro.

Eficacia vs Eficiencia

- **Eficacia**: Busca el cumplimiento de un objetivo, independientemente del número de recursos empleados. Tiene que ver con **<<Qué>>** cosas se hacen.
- **Eficiencia**: La eficiencia es hacer las cosas buscando la mejor relación posible entre los recursos empleados y los resultados obtenidos. Tiene que ver con **<<Cómo>>** se hacen las cosas.
- **Efectividad**: Se refiere a hacer las cosas de forma eficiente y eficaz. La efectividad tiene que ver con «qué» cosas se hacen y con «cómo» se hacen esas cosas.



Eficacia vs Eficiencia

La principal diferencia es que la eficiencia hace referencia a la utilización o empleo de un número menor de recursos; por otra parte la eficacia hace referencia a la capacidad para alcanzar un objetivo o meta de manera satisfactoria, aunque en el proceso no se hayan utilizado los recursos de manera óptima. Es decir, en la eficacia no importa si se logró ser eficiente durante el proceso para alcanzar un objetivo. Lo ideal sería ser eficaces y a la vez ser eficientes.

A la hora de construir software de calidad debemos ir en busca de Efectividad, es decir, Eficacia + Eficiencia.

¿Quiénes son responsables de la calidad del software?

La calidad del producto es responsabilidad de **todos los miembros del equipo** y debe procurarse en **cada etapa del desarrollo**, desde la definición de los requerimientos y estrategias de implementación y control, hasta su aplicación más directa en el código.

Para asegurar la calidad del software, debemos hacernos preguntas como las siguientes:

- ¿Están los requisitos plasmados de una manera adecuada y sin ambigüedades?
- ¿Existen criterios de aceptación establecidos? ¿Están escritos de forma precisa y medible?
- ¿Están definidos los parámetros de calidad que debe satisfacer el sistema?
- Para los bugs detectados, ¿se han informado al equipo de desarrollo? ¿Se ha abordado su corrección?
- ¿Se han realizado pruebas de robustez del sistema?
- ¿Se han realizado pruebas de estrés?
- ¿Se tienen claros los requerimientos no funcionales, y cómo se determinarán sus cumplimientos?

Entonces...¿por qué es importante construir software de calidad?

A continuación se presenta un resumen de los motivos por los cuales debemos enfocarnos en construir software de calidad.

- Para asegurar la satisfacción de nuestros clientes.
- Para prevenir riesgos futuros. Cabe resaltar que mientras más tarde se detecten los defectos o errores, mayores pueden ser las consecuencias.
- Para brindar soluciones claras a las necesidades de los usuarios.
- Para que nuestro software logre soportar todos los requerimientos, siendo amigable para el usuario, seguro, útil, usable, mantenible y estable.
- Para brindar exactitud y confiabilidad en los procesos, apoyándonos en cada una de las pruebas realizadas: funcionales y no funcionales.

2 | Calidad del código

Calidad del código

Los siguientes aspectos nos ayudarán a medir la calidad de nuestro código. Seguir buenas prácticas de programación y adecuarnos a los estándares nos permitirá construir código mantenible, seguro, robusto y de calidad.

- **Código duplicado** → Este término se utiliza cuando hablamos de un código fuente que aparece más de una vez, un buen desarrollo está asociado a la reutilización del mismo.
- **Código muerto** → Es el código que se encuentra en nuestra aplicación, pero nunca es utilizado. Normalmente aparece después de hacer una refactorización del código.
- **Estándares de codificación** → Se refiere a convenciones para escribir código fuente, las cuales frecuentemente son dependientes del lenguaje de programación. Las convenciones más comunes hacen referencia a: nombres de variables, indentaciones, espaciado, entre otros.

Calidad del código

- **Bugs** → Un bug es un error o un defecto en el software que hace que un programa funcione de forma incorrecta.
- **Complejidad ciclomática** → Es una métrica de calidad de software basada en el cálculo del número de caminos independientes que tiene nuestro código. Cuanto más compleja sea la lógica de un código, más difícil será de entender, mantener y probar.
- **Cobertura de código** → La cobertura de código es una medida que nos indica el porcentaje de código validado por los tests. Generalmente con una mayor cobertura aseguramos que no se introducen errores en una mayor parte del código, pero esto dependerá de la funcionalidad real que cubran los tests.

Herramientas para evaluar la calidad

Existen diferentes herramientas que nos permiten evaluar la calidad de nuestro código fuente, una de las más conocidas es **SonarQube**, este se trata de un software libre (en su versión community) que nos permite evaluar la calidad realizando un análisis estático del código, con el objetivo de advertirnos sobre diferentes puntos a mejorar.

Se denomina análisis estático del código al proceso de evaluar el software sin ejecutarlo. Pese a esto, es importante aclarar que si queremos comprobar la cobertura de los tests, estos sí deben ser ejecutados.

SonarQube

Podemos realizar análisis de código en más de 20 lenguajes de programación, entre ellos JAVA. Cada analizador proporciona numerosas reglas para detectar problemas de calidad generales y específicos de cada lenguaje.

SonarQube

Sonarqube usa diversas herramientas de análisis estático como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad, a través de las reglas que estas definen.

	Checkstyle	PMD	Findbugs
Propósito	Verificar el cumplimiento de las reglas de codificación.	Identificación de problemas potenciales.	Encontrar errores.
Tipos de verificación	- Convenciones de nombres - Encabezados, importaciones - Espacios en blanco, formateo - Comentarios de javadoc - Buenas prácticas, convenciones de códigos - Parámetros del método - Complejidad ciclomática - Cuialquier expresión regular	- Posibles errores - Código muerto - Código duplicado - Complejidad ciclomática - Expresiones sobrecomplicadas (legibilidad)	- Posibles errores - Defectos de diseño - Malas prácticas - Correción multiproceso (correcto manejo sincronía) - Vulnerabilidades de código

Issues y Reglas

Cuando un componente no cumple con una regla de codificación definidas por Checkstyle, PMD o Findbugs, se registra un issue.

Hay diferentes tipos de issues:

- **Bug:** un punto de fallo real o potencial en su software.
- **Vulnerability:** un agujero de seguridad que puede usarse para atacar su software.
- **Code Smells:** un problema relacionado con la mantenibilidad en el código. Los futuros desarrolladores tendrán más dificultades de las que deberían para realizar cambios en el código y esto puede provocar que introduzcan errores adicionales a medida que realizan cambios.

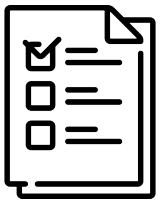
SonarQube

Quality Profile

Son colecciones de reglas que se revisan durante un análisis. Para cada tecnología hay un Quality Profile predeterminado, pero podemos tener varios para un mismo lenguaje.

En caso de que un proyecto analizado no tenga asignado explícitamente uno, se analizará con el perfil configurado como predeterminado para dicho lenguaje.

Quality Gate

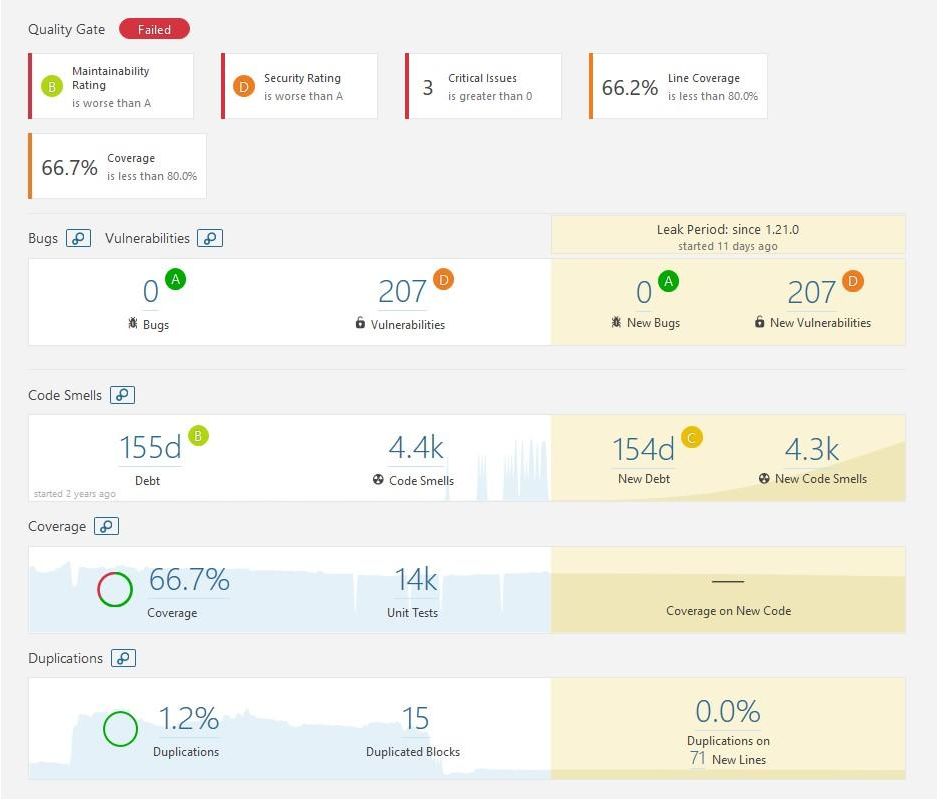


Es el conjunto de condiciones que el proyecto debe cumplir antes de que pueda ser lanzado a producción. Al igual que ocurre con los quality profiles, es posible establecer un Quality Gate predeterminado.

Siempre que se pueda está bien tener un mismo quality gate y quality profile para todos los proyectos de la empresa. Esto nos garantiza un estándar de calidad dentro de la compañía y también hace más liviana la movilidad de los desarrolladores entre proyectos.

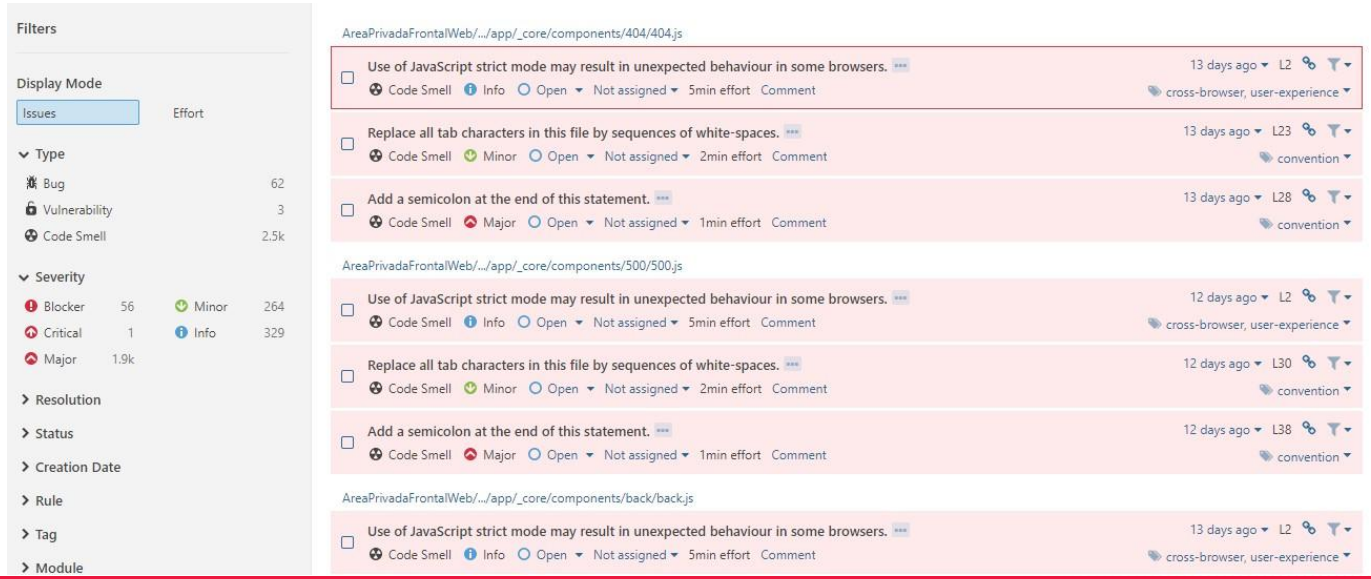
Vista del proyecto

Cada proyecto dispondrá de una pantalla con el resumen del mismo, la cual se actualizará después de cada análisis. Desde esta vista podemos observar información referente a: bugs, vulnerabilidades, code smells, cobertura, duplicidad, ejecuciones, quality profile y quality gate asignado.



Vista de issues

Desde esta vista podemos gestionar y obtener más detalles sobre los issues pendientes. Pulsando en cada una de ellas podemos ver la línea exacta donde se genera el mismo, una breve descripción, e incluso se nos ofrecerá una posible solución.



SonarQube

Las presentadas son sólo algunas de las características de SonarQube, para más información visita la documentación oficial del proyecto:

+ info

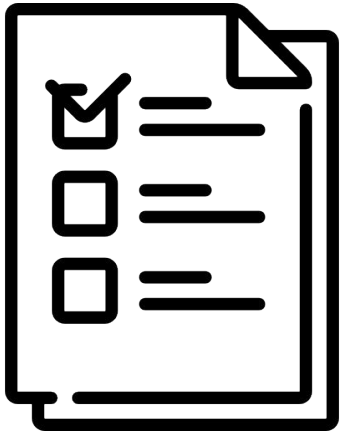
Enlace para visitar:

SonarQube
Documentación oficial.

3 Testing Automatizado vs Testing Manual

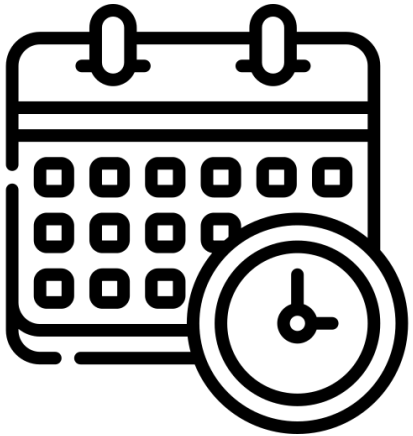
Testing Manual

Las pruebas manuales son ejecutadas por uno o más testers que, partiendo de casos de prueba, simulan las acciones del usuario final con el fin de detectar fallas y reportarlas. Las pruebas manuales son utilizadas principalmente para pruebas funcionales, de usabilidad y/o exploratorias.



Testing Automatizado

Las pruebas automatizadas son ejecutadas por testers con skills técnicos, y se apoyan de diversas herramientas para realizar scripts y así, ejecutar las pruebas automáticamente y de manera repetitiva. Al eliminar las tareas rutinarias, este método permite liberar tiempo para realizar acciones más profundas vinculadas con el desarrollo e implementación de software.



Pruebas Manuales

Son indispensables para cada proyecto, ya que por medio de ellas se detectan la mayor parte de los defectos, y de esta forma se logra estabilizar el sistema que se está probando.

Permiten un análisis profundo por parte de un humano, lo cual es un mayor beneficio cuando se quiere mejorar la experiencia de usuario.

Pruebas Manuales

La repetición de pruebas manuales podrían dejar pasar defectos, ya que una acción repetitiva para una persona puede tornarse cansadora y tediosa.

Las pruebas de regresión manuales pueden llegar a consumir mayor tiempo que las automatizadas.

Pruebas Automatizadas

Pueden ejecutarse una y otra vez luego de ser creadas, y de manera más rápida que las pruebas manuales. Son ideales para ampliaciones de funciones y verificando que lo que no se ha modificado siga funcionando.

Son la opción más viable cuando se necesita ejecutar diversos test cases, de manera repetitiva y por un período de tiempo extenso.

Pruebas Automatizadas

Pueden brindar un mayor grado de confiabilidad cuando se aplican después de algún cambio al hardware o software donde se ejecuta el sistema.

La automatización no es un reemplazo del testing manual sino un proceso complementario.

DigitalHouse>