

¿Cuándo se automatiza y no?

1. Triple Restricción

- Tiempo: si no hay tiempo no podemos automatizar, ya que se la automatización consume más tiempo que las manuales crear librerías, arquetipos, clases genéricas
- Recursos: Si no hay personal o plata
- Alcance : sea automatizable, que tenga adherencia mínima, que tenga una herramienta que puede hacer esa adherencia

2. Viabilidad técnica va pegada al alcance que el aplicativo sea mínimamente estable, que no esté cambiando, que esté controlado los cambios y que obedezcan a un requisito.

Que el robot tenga posibilidad de regresión (que tiene tiempo, recurso y el alcance)

Si solo se va a ejecutar una vez no sirve, para que eso.

Motivo de negocio

3-Que tenga una criticidad, prioridad alta que sirva para algo porque si tenemos las triple restricción y no sirve para nada

criticidad: es que si falla se tiene un riesgo alto

4- Que la prueba se pueda auto validarse, que se pueda comprobar, que se pueda verificar y validar en otro lado. para poder que el robot sea independiente

¿que es automatización?

La automatización es una técnica de prueba que se apoya en las herramientas de software para generar una ejecución automática de una prueba de software reemplazando la labor humana

Proceso de pruebas :

Si arrancas desde 0, arrancas con el entendimiento de los requerimientos, luego el diseño o creación del plan de pruebas, dentro del cual definen la matriz de riesgos, y aportas ideas para la mitigación de los mismos, también los criterios de inicio, pausa y finalización de las pruebas. Luego llega el diseño y escritura de los escenarios, después la ejecución, toma de evidencias e informes de pruebas. Y finalmente el cierre del proceso.

Traduciendo esto a algo diario, es, el entendimiento de la necesidad del cliente, el diseño y ejecución de los test cases, el informe diario en tus Daily, la toma de evidencia en La herramienta de tracking que utilicen, las review de cada sprint y por último el despliegue del proyecto.

Diferencia entre el tester y el qa

El Testing es un proceso de ejecución del sistema que trata de encontrar y reportar fallos a través de distintos casos de prueba. En cuanto al Quality Assurance, se trata de un conjunto de actividades que logran medir la calidad del producto, además de establecer procesos y un

mantenimiento adecuado para garantizar que el sistema cumpla con las expectativas de producción.

Temas generales de pruebas:

- Niveles de pruebas (unitarias, integración, sistema , aceptación)

Unitarias:

Validamos que las unidades construidas cumplan con lo establecido

Integración (se prueban el backend servicios microservicios,,apis, mensajería de colas)

Es donde podemos validar que todo esté en orden y que todo esté integrado con el resto de componentes

un micro servicio que hacen una sola función cosa ejemplo la suma, resta, división y multiplicación

un servicio es la calculadora

una api es un software matemático ejemplo matlab tiene

mensajería de colas es cuando los componente de frontend se comunican con los de backend por medio de colas y mensajes

No meterme con esto:

(Los buses son transportadores de varias colas

broker son transportadores de varios buses)

Sistema:

Si ya todo está integrado es un sistema, donde validamos (si se está construyendo el producto correcto que es lo que el usuario realmente quiere) y verificamos (si se está construyendo de forma correcta) el sistema

Aceptación:

Es una prueba acorde a los requisitos establecidos por los usuarios y es donde el tester y el usuario deciden si aceptan o no el desarrollo y se cierra con una reunión UAT

User acceptance testing(uat): después de terminado un usuario hace un criterio final basado en lo que uno le dice.

Beneficios de automatizar:

- Asegurar regresiones
- ejecuciones gratuitas
- tenemos controlado el control de cambios
- componentes genéricos que podemos reutilizar
- Reportes dinámicamente

-Puedo generar metricas e indicadores

indicador: ahorro tiempo y plata

ejemplo pruebas manuales vs pruebas de automatización

el tester se demora una hora por cada prueba cuando ya lleva 10 horas el automatizador apenas ha creado el script pero cuando pase a la onceava prueba la prueba de automatización cada segundo corre varias pruebas a cambio la prueba manual se sigue demorando una hora.

en la prueba manual es mas barato pero se demora mas, en la automatizador es mas caro pero en la corrida numero 10 cuando llega a la 11 se demora menos llega y se encuentra un punto de retorno donde ya es más barato ejecutar una prueba de automatización que hacer una a mano

y esto se materializa en el ROI RETORNO DE INVERSIÓN

métricas:

que cobertura tiene si tiene 100 casos y solo cubrió 25 solo tiene 25% cubierto

indicador de errores: los reportes de serenity

naturaleza de errores: podemos identificar la naturaleza de los errores si fallan 30 de 100.

- Tipos de prueba (funcionales – no funcionales)
 1. Funcionales: (podemos aplicar aceptación,smoke test o componentes)
que el software que haga lo que tenga que hacer
Las pruebas funcionales son aquellas que se centra en probar la funcionalidad del sistema
 - Adecuación
 - Exactitud
 - Interoperabilidad
 - Seguridad
 - Cumplimiento de funcionalidad
 2. No-funcionales (performance):
Evalúan Características del sistema y del software
 - Fiable: madures a fallos, recuperabilidad
 - Usabilidad: fácil de utilizar
 - Eficiencia: carga y estrés
 - Mantenibilidad: que se pueda escalar y mantener a futuro
 - Portabilidad: fácil de instalar
 3. Pruebas asociadas al cambio

Es probar cuando el software ha sufrido algún cambio , estos cambios pueden ser después de detectar y corregir un issue

- Pruebas de regresión: cuando se hace una corrección y miramos que no afecto ninguna de otras funciones
- Prueba de confirmación: es donde probamos nuevamente para ver si no se presente el issue
- TDD (desarrollo orientado por pruebas)
Escribir primero las pruebas unitarias y después escribimos el código fuente y después refactorizar el código escrito
Y para hacer una buena práctica de TDD podemos guiarnos con los principios FIRST
 - Fast (rápido)
 - Independent (independiente) Que las pruebas no dependan de las otras
 - Repeatable (repetible) ejemplo del servidor funciona en local pero en el servidor no
 - Self-validating (auto evaluable): Uno de los puntos a favor de pruebas automatizadas es que podemos ejecutarlas simplemente al pulsar un botón o incluso hacer que se ejecuten de forma automática tras otro proceso, como podría ser un push a una rama.
 - Timely (oportuno).las pruebas deben desarrollarse antes de empezar a desarrollar el código
- BDD (desarrollo orientado por comportamientos – gherkin)
Behaviour Driven Development
Es donde se utiliza el lenguaje gherkins que es donde colocamos los criterios de aceptación que es los requisitos que necesita el cliente
- ATDD Ver el siguiente Link
en el equipo se dividen los criterios de aceptación y luego se dividen el conjunto de pruebas de aceptación antes de que comience desarrollo
- Técnicas de diseño
 - **Caja negra**
 - Partición de equivalencia
 - valores límites
 - tablas decision
 - transición de estados
 - Pruebas de casos de uso:
 1. Caso de prueba
 - ID
 - PRIORIDAD
 - REQUISITO
 - PRECONDICIONES
 - VALORES DE ENTRADA
 - PASOS
 - RESULTADO ESPERADO
 2. Caso de uso
 - Actores
 - Propósito

- Resumen
 - Requisito
 - Precondición
 - postcondición
- **CAJA BLANCA:**
 - Cobertura de sentencia
 - Cobertura de decisión
 - Todos los pares
- **Técnicas basadas en la experiencia**
 - Pruebas exploratorias: habilidad y conocimiento del tester para guiar las pruebas
 - Predicción de errores: se utiliza el conocimiento del tester de cómo se comportó la aplicación en el pasado
 - checklist
- Técnicas de estimación de pruebas
 1. Basada en métricas (Metrics-based approach)

Se analiza información de datos y métricas de proyectos anteriores

Se utiliza datos recopilados como:

números de condiciones de prueba,

casos de prueba diseñados y ejecutados,

issues,

tiempo de diseño y ejecución

puntuando las historias de usuario para saber cuánto nos demoramos
 2. Basada en expertos (expert based)

Consiste en consultar a las personas que harán el trabajo, las que tienen más experiencia en las tareas que se van a realizar

(tester,arqui,consultores)
- Caso práctico

Se plantea por parte del entrevistador una situación de la vida real, donde el entrevistado debe explicar en términos de prueba cual sería su estrategia y plan de pruebas de cara a solucionar dicha situación.

Casos propuestos: envío de un correo de gmail con su respectiva respuesta por parte del receptor

Ir a una cafetería a comprar algo con cierto presupuesto
- Bugs:

Nos van a preguntar: Que es un bug, que contiene y como se reporta

bug :

se debe tener un título descriptivo

una descripción

un paso a paso

ambiente donde se hizo la prueba

el resultado obtenido y el resultado esperado

el estado en el que va a quedar

y ellos tienen que dar un tiempo de respuesta para hacer las mediciones para empezar la prueba
evidencia

Temas generales de programación

- Normalmente se pregunta por los componentes de un proyecto de java, es decir:
- Arquetipos
un arquetipo es un patrón o modelo sobre el que se pueden desarrollar todas aquellas tareas que son de un mismo tipo
- Que son y para qué sirven
Para reutilizar y estandarizar los proyectos de una organización
- Clases en java, objetos
objeto tiene funciones o variables y pertenecen a cierto objeto
métodos y atributos

clase se pueden crear instancias de objetos
con atributos independientes

encapsulación: (encapsular los valores)
se encarga de ocultar los atributos y métodos en la cual solo puede ser accedido desde la misma clase
tenemos que utilizar los métodos accesorios (getter y setter para acceder atributos de un objeto)

set establece la edad

get mostrar la edad

abstracción se tiene que heredar de la clase padre con el extends
sabemos cómo interactuar con la clase pero no sabemos cómo funciona por dentro
un método abstracto va a estar en una super clase padre, pero no sabemos cómo se implementa

se puede implementar el método abstracto en subclases que también tengan hijos.

No se puede instanciar objetos

el método no se implementa solo se coloca ; y si el método es abstracto la clase también debe de serlo.

en la clase hijo se sobreescribe los métodos @override

Herencia

reutilización de atributos y métodos comunes

utilizamos constructor para inicializar los atributos, después utilizamos los get para cada atributo que está en el constructor.

para heredar es con extends

utilizamos super para utilizar los parámetros de la clase padre, cuando estamos en la subclase y queremos llamar esos parámetros.

polimorfismo

muchas formas que pueden tomar un objeto
en una relación de tipo herencia, un objeto de la superclase puede almacenar un objeto de cualquiera de sus subclases.

constructor y getters

un objeto tiene atributos(características)

por ejemplo un carro es un objeto y tiene atributos como color, marca, km y los **métodos**(acciones) que es encender, acelerar y frenar

los objetos pueden ser reales o ficticio:

Ejemplo

3/2

atributos:

denominador y numerador

acción:

simplificarse

sumar o restar con otra fracción

Clases:

un conjunto de objetos con características similares

Se inicializa con mayúscula

- Método en java
Son acciones
- Librerías
Es un conjunto de clases, que poseen atributos y métodos
- Jars
es un tipo de archivo que permite ejecutar aplicaciones y herramientas escritas en el lenguaje Java

- Dependencias
Cuando una clase depende de otra
- métodos de acceso
private, public, protected

Agilismo

- Metodologías ágiles
Qué son y cuales existen, se hace énfasis en scrum y kanban

un conjunto de buenas prácticas para **trabajar en equipo**, y obtener el mejor resultado posible de un proyecto
está especialmente para proyectos en **entornos complejos**, donde se necesita **obtener resultados pronto**
un proyecto se ejecuta en ciclos temporales cortos y de duración fija de 3 a 4 semanas.

Es un marco de trabajo

Planificación de la iteración:

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

1. **Selección de requisitos** (2 horas). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que prevé que podrá completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.
2. **Planificación de la iteración** (2 horas). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos seleccionados. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se auto asignan las tareas, se autoorganizan para trabajar incluso en parejas (o grupos mayores) con el fin de compartir conocimiento (creando un equipo más resiliente) o para resolver juntos objetivos especialmente complejos.

Ejecución de la iteración

Cada día el equipo realiza una **reunión de sincronización** (15 minutos)

Para decir que he hecho, que voy hacer y que impedimentos tengo

- Ceremonias ágiles SCRUM
Cuales son:
(Daily (se hace diario 15 minutos)
inception(contextualización ANTES de iniciar proyecto)
review (al finalizar el sprint)

planning (dividir trabajo al inicio de cada sprint) etc...)

Retrospectiva: es una review a nivel personal

- Roles de agilísimo

(PO (jefe del negocio representa al cliente y conoce la necesidad del cliente y trata de transmitir al scrum como al team dev para construir esa necesidad), BA (analista de negocio), Scrum master (modera y ayuda al team dev ,dirigir el equipo), Development team(desarrolladores y devops Qa's etc...)

El PO define un product backlog (lista de necesidades del cliente) se lo manifiesta al scrum y al team dev en una reunión que se llama sprint planning meeting donde planeamos cómo le damos solución a esta primer fase donde seleccionamos las tareas que se van a trabajar en el sprint. Obtenemos una lista de tareas que se va a trabajar en el sprint esto se llama sprint backlog. Que son requisitos que se van a construir de 1 a 4 semanas.

Se hacen dailys (15 min) para decir que he hecho, que voy hacer y qué impedimentos tengo

Se utiliza tablero tipo canvan do, doing ,done

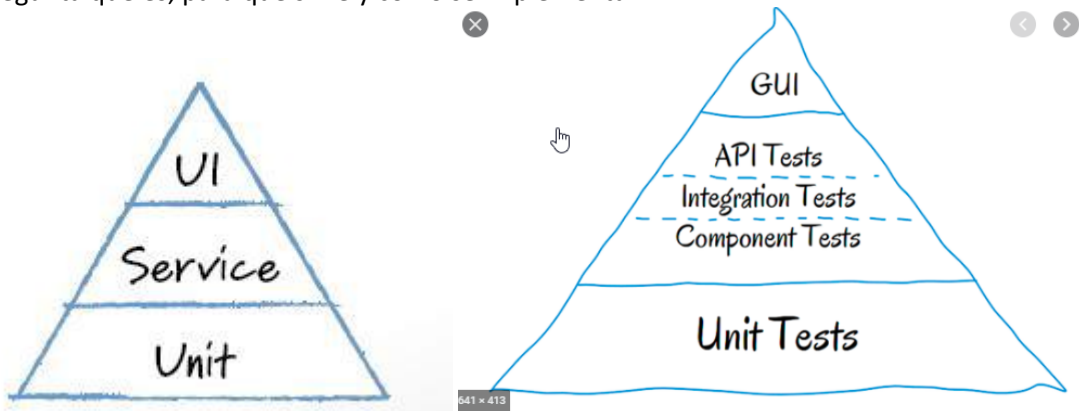
Si un compañero termina trato de ayudarle a otro compañero

Cuando finalizamos el sprint hacemos la **review para verificar el cumplimiento de las metas** y así garantizamos la entrega del producto al cliente final

Retrospectiva para mirar si encontramos alguna falencia en el proceso o mejoras para el siguiente sprint y volvemos al product backlog

Pirámide de Cohn (muy importante)

Se pregunta qué es, para que sirve y como se implementa



- Pruebas unitarias (pruebas hechas por el desarrollador en cada componente)
(Qué son y que herramientas existen para ello: protractor, Jasmine, junit)
- Pruebas de integración (hechas por el QA, prueban integración de servicios, microservicios) – soap vs rest
(Qué son y qué herramientas existen para ello: postman, rest assured, soap ui etc...)

- Pruebas de interfaz gráfica GUI (probar por robot lo que se ve en cada pantalla según transacciones)
(son las pruebas que normalmente hacemos automatizadas)

Principios y patrones de programación de pruebas:

- **SOLID**
Es uno de los principios que rige la programación orientada a objetos
(1. Que una clase haga solo una cosa 2. que la clase sea abierta pero cerrada (usar pero no modificar) 3. que un método de una clase no afecte a métodos de la clase madre) son 5 conceptos en total

Solid

que sea de código mantenible, fácil de hacer cambios y que sea muy extensible.

Single responsibility :

cada clase debe de tener una sola responsabilidad y debe de encargarse de una sola parte del sistema

que la clase solo haga una sola cosa.

Open/Closed Principle (OCP)

una entidad debe de quedar abierta para su extensión pero cerrada para su modificación

la funcionalidad esté protegida, que sea difícil romper para añadir funcionalidades tenemos que escribir nuevo código y no modificar el código existente que ya funciona

la herencia y el polimorfismo

Liskov Substitution Principle (LSP)

es que toda clase de que es hija de otra clase debe poder utilizarse como si fuera el mismo padre.

ejemplo una clase pato que tiene métodos nadar, volar y hacer cuack pero hay un pato de goma que no vuela y se devolverá un error ya que no vuela entonces la solución es rediseñar el sistema que el pato de goma no herede de pato sino que implementa las interfaces que se necesite

Interface Segregation Principle (ISP)

es que es mejor tener muchas clases pequeñas y especializadas en una sola que tener una clase con muchas funciones

explicación de la impresora que hacía de todo, grapar y mandar fax pero al momento de programar todo estaba en una sola clase y un cambio insignificante afectaba mucho el tiempo de respuesta del programa

Dependency Inversion Principle (DIP)

los módulos de altos nivel no debería de depender de los módulos de bajo nivel ambos deberían depender de interfaces.

por ejemplo si nuestro programa necesita extraer la info de una b.d no tiene que depender si es de una tecnología u otra, ya que nuestro código no depende de qué base utilizamos sino de la abstracción que construimos para sacar esa data, entonces si nuestro programa podemos cambiar sin ningún problema si utilizamos mongodb a ponerle sql sin afectar ninguna parte del sistema

- **FIRST (Muy Importante)**

(Rápidas - independientes - repetibles - auto validadas - a tiempo)

Fast

Una de las ventajas que nos ofrecen los test unitarios es la posibilidad de ejecutar un gran número de tests en cuestión de segundos.

Independent

Por muchas pruebas unitarias que tengamos, todas deben de ser independientes de las otras.

Repeatable

El resultado de las pruebas debe ser el mismo independientemente del servidor en el que se ejecute.

SELF-VALIDATING

Uno de los puntos a favor de pruebas automatizadas es que podemos ejecutarlas simplemente al pulsar un botón

Timely

deberíamos tener desarrolladas las pruebas, que deben estar desarrolladas lo antes posible y siempre antes de subir código a producción.

diferencia entre pom y screenplay

pom orientado a los requisitos del usuario y screenplay orientado a tareas del negocio

- **POM**

Modelo o PATRON para automatizar pruebas BDD, las 5 capas :

diseño(gherkin)

- definition (pasa el lenguaje natural a código fuente)
- Steps (coordina la ejecución de los pasos y con que lógica)
- Page (hago la iteración del software con el aplicativo y debería de ser independiente para que se pueda utilizar)
- Ejecucion

Es el modelo por el medio del cual se convierte en programación la metodología BDD (las capas aquellas Feature, Definition, Steps, Pages)

- PageFactory

Es otro modelo de programación de pruebas NO usado por sura pero posiblemente implementable por el cliente

El patrón PageFactory se utiliza, junto con el patrón Page Object a la hora de implementar pruebas funcionales para hacer el código más mantenible al momento de capturar los xpath no almacenarlos en un driver.findelement si no en un @findby

Y después inicializamos las clases con el driver y la propia clase

Inicializar esos elementos de la clase (u objeto de Page Object) con el PageFactory: (hay que pasar el driver y la propia clase)

```
public Search(WebDriver d2){  
    driver = d2;  
    PageFactory.initElements(driver, this);  
}
```

Y realizar operaciones sobre estos elementos en el método pertinente de la siguiente forma:

```
public void searchString(String product){  
    driver.get("https://www.amazon.com");  
    searchInput.sendKeys(product); //p.e enviar un string a ese input  
}
```

- Screenplay donde se necesite un alto de reutilización de componentes
good testing habits and well-designed test suites by enabling teams to write more robust and reliable tests

feature

stepsdefinition

runner

question son los assert

iteration son las acciones

Temas de integración continua

- Integración continua

Se crea las dependencias y después ejecutamos el test

(proceso para hacer integraciones automáticas de un proyecto en un repositorio versionado)

- Despliegue continuo

(promoción de las versiones integradas YA PROBADAS entre ambientes hasta llegar a producción)

- Diferencia entre ambos términos (Muy importante)

Temas de DevOps

- **Que es devops (continuous integration and continuous delivery)**

Comunicación, colaboración y integración entre desarrolladores y operadores
(conjunto de herramientas y prácticas para hacer integración y despliegue continuo, que las dependencias no tengan vulnerabilidad que se cumpla un estándar de seguridad)
Es un marco de trabajo para que se cumpla con las metas y

es un tratar de conectar operadores con developers porque las metas se alinean, la comunicación. velocidad y resultados

si yo soy un operador tengo que mantener el código corriendo, entonces el desarrollador me tiene que dar guías de que va a salir bien y como lo puedo arreglar y que va a salir bien y como lo puedo observar, como puedo buscar errores como vas a entender el servicio, yo como desarrollador puedo ayudar

devops es mas:

- cultura- el deseo en ambos equipos en colaborar
- automatización - reproducir infraestructura como reproducioms bugs
- medidas- saber cuanto nos demoramos en cumplir nuestras metas

el código llega a producción con menos errores ya que la comunicación es lo que es mas primordial entre operadores y desarrolladores

- **Que es Análisis estático de código fuente**

(validación de buenas practicas en el código fuente escrito antes de ser desplegado)

- Herramientas para análisis estático (sonarQube)

- Que es Git

(REPOSITORIO y controlador de versiones)

- Que es Gitflow

(conjunto de comandos que me permiten interactuar con los repositorios)

- Diferencia entre ambos (Git vs Gitflow)

- Que es SonarQube y cómo funciona

(es la herramienta que me permite hacer análisis estático)

- Que es Jenkins

Herramienta que dispara tareas de integración, compilación, ejecución de pruebas e impresión de evidencias admite uso de pipelines en los ambientes correspondientes