

Informe de investigación sobre los temas abordados en clase



Célula de trabajo Gryffindor

Docente de la asignatura

ROBINSON CORONADO GARCIA

Introducción

1. Cuatro más una vista
 - Vista lógica
 - Vista de procesos
 - Vista física
 - Vista de despliegue
 - Escenarios
2. Jmeter
3. Programación reactiva
4. Diseño Dirigido por el dominio (DDD)
5. Algunos requisitos no funcionales
 - Certificación
 - Conformidad
 - Gestión de configuración
 - Testabilidad
 - Transparencia
 - Capacidad - actual y pronosticada
 - Durabilidad
 - Documentación
 - Recuperación de desastres
6. Referencias

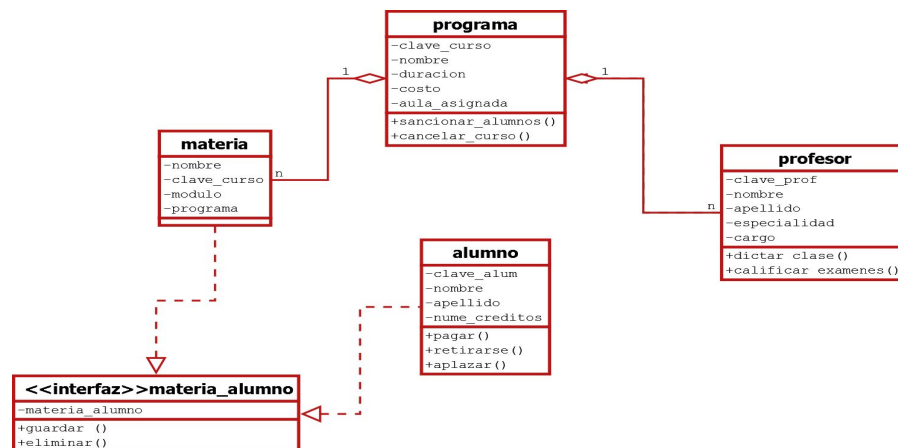
1. CUATRO MÁS UNA VISTA

Es un modelo que sirve para describir la arquitectura de sistemas software basándose en el uso de múltiples vistas concurrentes.

Básicamente es abordar varias capas de un sistema, a partir de modelos y diagramas para luego unificarlos en un modelo común o general.

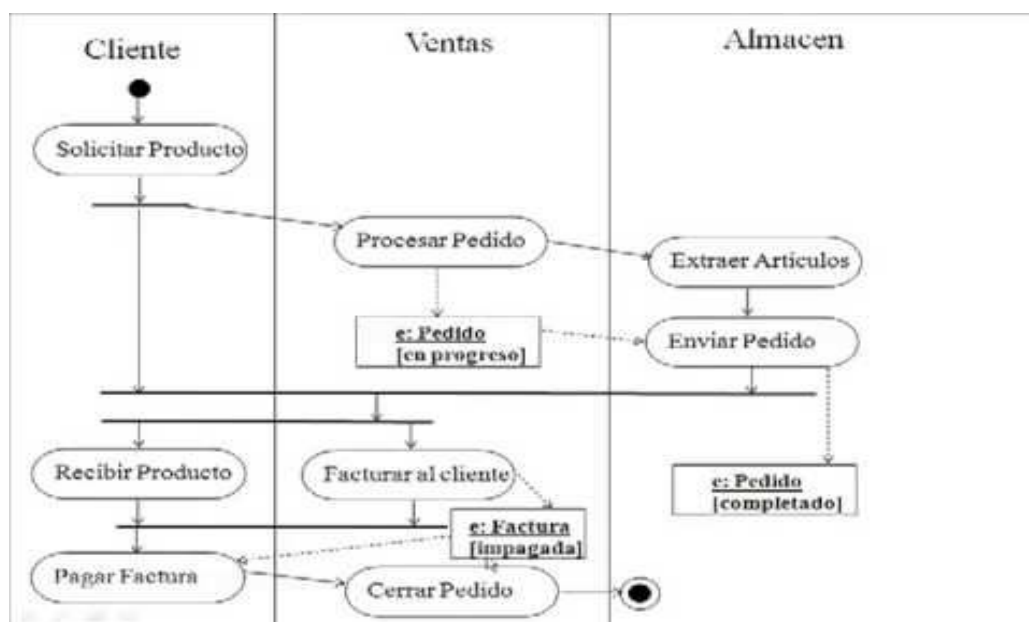
1.1 Vista lógica:

En esta vista se describe la estructura y funcionalidad del sistemas, está directamente relacionada con un diagrama de clases o de secuencia.

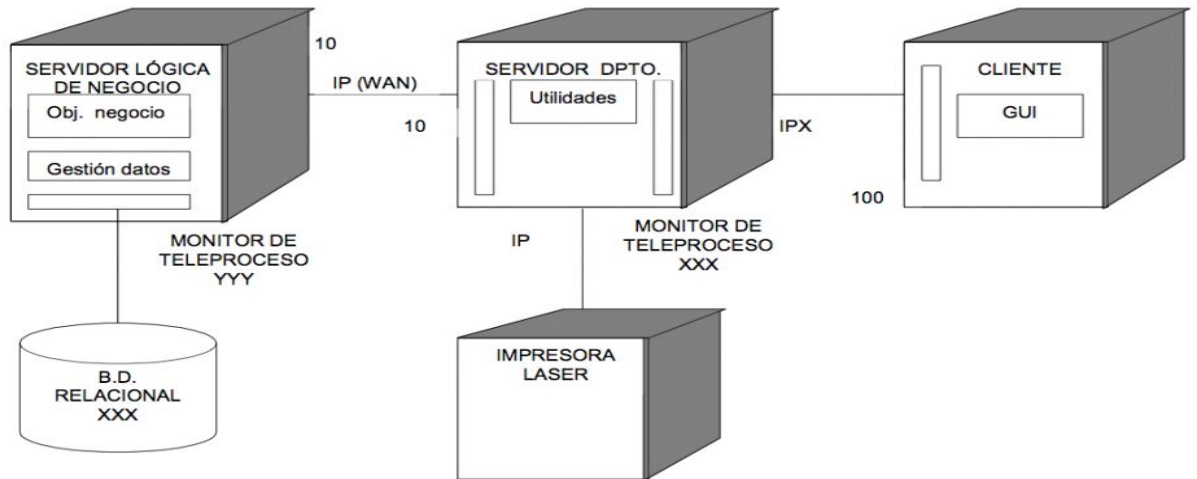


1.2 Vista de procesos:

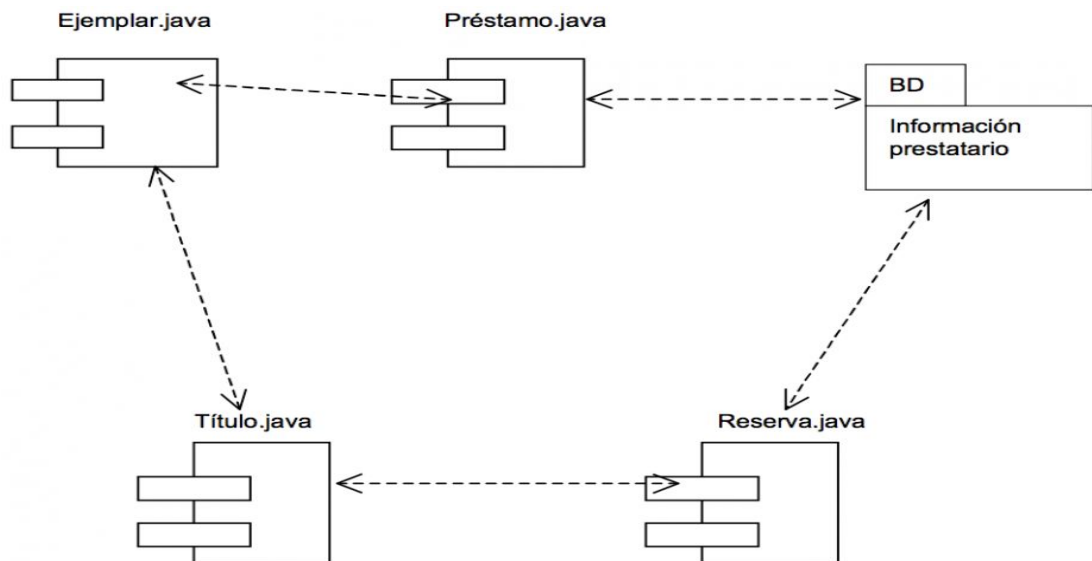
Se muestran los procesos o el conjunto de actividades de un sistemas, y cómo se comunican entre ellas, en general muestra el flujo de trabajo de todo el proceso, está directamente relacionada con un diagrama de actividades o de procesos.



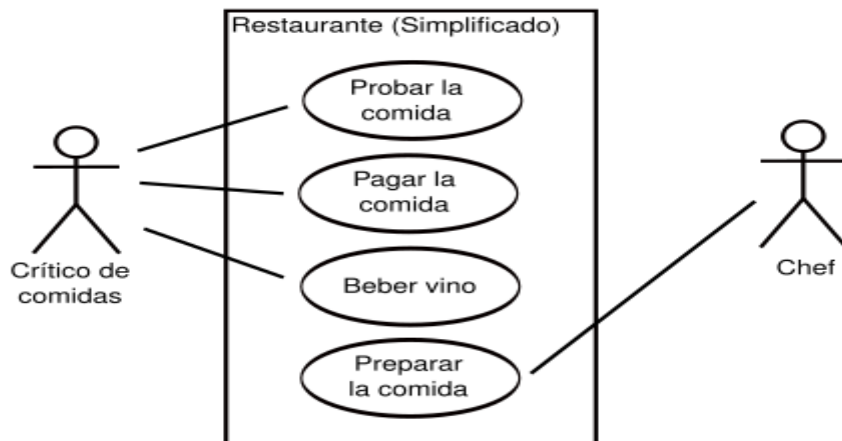
1.3 Vista física: Es la vista de un ingeniero de sistemas, aquí se muestran todos los componentes físicos del sistema y las conexiones de estos. Se relaciona con el diagrama de despliegue.



1.4 Vista de despliegue: Acá se muestra la vista de un programador y básicamente muestra cómo está dividido el software en componentes. Se relaciona con el diagrama de paquetes y de componentes.



1.5 Escenarios (Vista +1): Esta vista es la que se representa por medio de los casos de uso y tiene la función de unir y relacionar a las otras 4 vistas. con lo que al final tendremos una trazabilidad de componentes, clases, equipos, paquetes, etc.

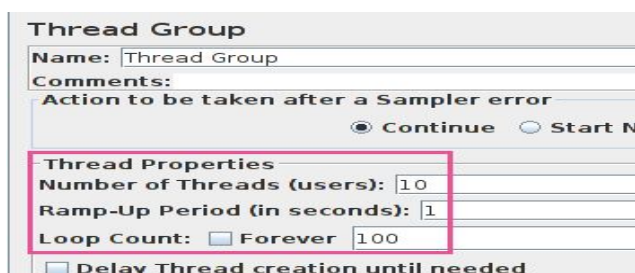


2. JMETER

En resumen es una herramienta que nos permite a hacer tests o pruebas, generalmente para medir el desempeño de servicios, conexiones entre otras.

Inicialmente diseñada para pruebas de estrés en aplicaciones web, hoy en día, su arquitectura ha evolucionado no sólo para llevar a cabo pruebas en componentes habilitados en Internet (HTTP), sino además en Bases de Datos , programas en Perl , requisiciones FTP y prácticamente cualquier otro medio.

Normalmente se delimitan la concurrencia con la que se va a grabar el test.



luego se configuran todas las conexiones y puertos con los que se va a hacer la prueba.

Configurar acceso proxy a Internet

☐ Sin proxy

☐ Autodetectar configuración del proxy para esta red

☐ Usar la configuración del proxy del sistema

☒ Configuración manual del proxy

Proxy HTTP: 192.168.2.91 Puerto: 8181

☐ Usar el mismo proxy para todo

Proxy SSL: Puerto: 0

Proxy FTP: Puerto: 0

Host SOCKS: Puerto: 0

☐ SOCKS v4 ☒ SOCKS v5

No usar proxy para:

Después de estar haciendo la prueba, por medio de *Listeners*, se pueden ver los datos que se están esperando probar y ver su comportamiento.

Summary Report										
Name: Summary Report										
Comments:										
Write results to file / Read from file										
Filename										
		Browse...		Log/Display Only: <input type="checkbox"/> Errors <input type="checkbox"/> Successes		Configure				
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB...	Sent KB/sec	Avg. Bytes
1 /	2000	242	9	650	230,94	0,00%	41,2/sec	1343,39	79,81	33354,2
2 /reserve.p...	1000	65	56	141	13,50	0,00%	20,8/sec	216,40	9,64	10658,9
3 /purchase...	1000	65	56	132	10,49	0,00%	20,8/sec	202,78	10,84	9973,0
4 /confirmat...	1000	64	56	133	9,48	0,00%	20,8/sec	182,97	12,16	8995,9
5 /home	1000	133	114	232	17,71	0,00%	20,8/sec	335,88	15,18	16533,3
6 /index.php	1000	63	56	184	10,20	0,00%	20,8/sec	161,82	7,37	7949,0
7 /register	1000	66	57	129	10,89	0,00%	20,8/sec	224,42	7,45	11022,1
TOTAL	8000	117	9	650	138,14	0,00%	165,0/sec	2655,04	141,85	16480,1

Finalmente con estos datos, se sacan las conclusiones necesarias para tomar decisiones con respecto al software que se quería probar.

3. PROGRAMACIÓN REACTIVA

La programación reactiva es un paradigma de programación del que probablemente se escuche hablar actualmente pero que seguramente será muy común en un futuro, el hecho de que sea un paradigma por sí mismo no significa que no se traslape con otros paradigmas, el paradigma reactivo tiene dos objetivos principales, el primero es que debe propagar los cambios que ocurran dentro de las variables del sistema de manera rápida y sin mucho esfuerzo y el segundo que es el de trabajar con flujos asíncronos de datos, veamos algunos ejemplos, para el caso de la propagación de cambios imaginemos una hoja de cálculo en la que tomas la celda C1 y le colocamos una fórmula: $SUM=A1+B1$ que dependa de los valores de A1 Y B1, luego de colocar los valores en las dos celdas

referenciadas el valor de la celda con la fórmula automáticamente cambia, haciendo que efectivamente los cambios se propaguen en nuestro sistema como se observa en la figura1.

	A	B	C
1			=A1+B1
2			

	A	B	C
1	2	5	7
2			

(Figura 1)

Para el segundo caso de los flujos asíncronos de datos imaginemos una hoja en blanco en el procesador de texto, cada vez que presionas una tecla estás enviando un dato, entonces el procesador de texto reacciona de acuerdo a como fue programado para hacerlo en este caso colocar la letra donde corresponde, este flujo es asíncrono por que el procesador de texto no tiene certeza de la siguiente letra que se va a ingresar.

La programación reactiva es una programación que nos ayuda a lidiar con flujos asíncronos de datos, volviendo al ejemplo de la hoja de excel podemos considerar que cada nuevo valor dentro de una celda es solamente un dato nuevo proveniente del flujo de datos, proveniente del flujo de la información que provee el usuario o una fuente externa.

Si definimos el código de la figura 2 dentro de un esquema de programación reactiva el valor resultante de la variable C al final será 10 puesto que C establece una relación entre A y B , cualquier cambio que ocurra en cualquier de estas dos variables se verá reflejado en C

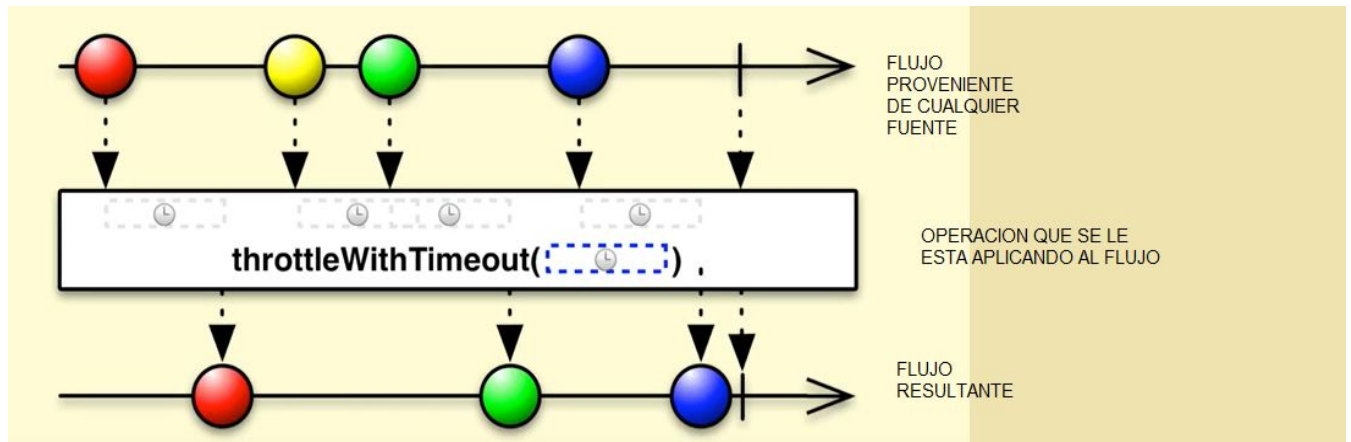
```
var a = 2;  
var b = 5;  
var c = a + b;  
a = 5;  
Console.Write(c);
```

(Figura 2)

Con este ejemplo podemos notar que la programación reactiva es una forma de implementar el patrón del observador

PATRÓN DEL OBSERVADOR CON PROGRAMACIÓN REACTIVA

Observer (patrón de diseño) Observador (en inglés: Observer) es un patrón de diseño de software que define una dependencia del tipo uno a muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.



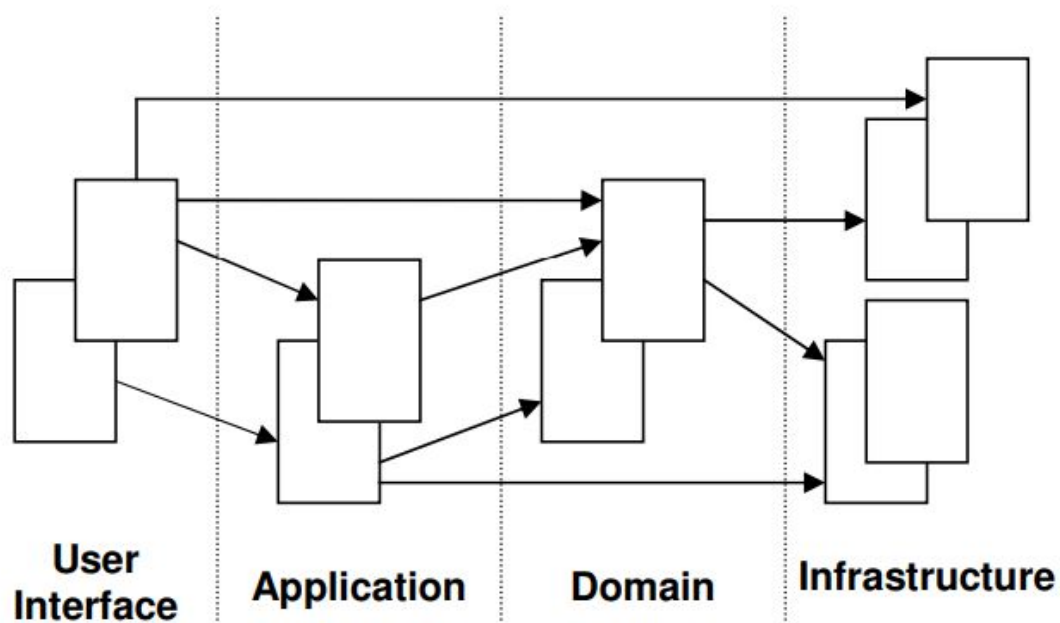
(Figura 3)

La programación reactiva es un paradigma enfocado en el trabajo con flujos de datos finitos o infinitos de manera asíncrona. Su concepción y evolución ha ido ligada a la publicación del Reactive Manifesto, que establecía las bases de los sistemas reactivos, los cuales deben ser:

- Responsivos: aseguran la calidad del servicio cumpliendo unos tiempos de respuesta establecidos.
- Resilientes: se mantienen responsivos incluso cuando se enfrentan a situaciones de error.
- Elásticos: se mantienen responsivos incluso ante aumentos en la carga de trabajo.
- Orientados a mensajes: minimizan el acoplamiento entre componentes al establecer interacciones basadas en el intercambio de mensajes de manera asíncrona.

4. DISEÑO DIRIGIDO POR EL DOMINIO(DDD)

Es una técnica estructurada por varias prácticas que nos podrán ayudar a tomar decisiones de diseño con el fin de enfocar y acelerar el manejo de dominios complejos.



DDD se basa en los siguientes puntos:

- Colocar modelos y reglas de negocio de la organización, en el core de la aplicación.
- Basar nuestro dominio complejo, en un modelo de software
- Se utiliza para tener una mejor perspectiva a nivel de colaboración entre expertos del dominio y los desarrolladores, para concebir un software con los objetivos bien claros.

Hay distintos aspectos que pueden generar la caída de un proyecto, objetivos pocos claros, problemas técnicos del equipo, problemas de elicitación de requerimientos; pero en software de un dominio complejo y una esperanza de vida un poco más larga puede ser que su caída sea debido a problemas de diseño.

Pero lo que realmente resuelva el DDD dependerá mucho del caso, nos podemos encontrar con la complejidad de muchas aplicaciones no está en la parte técnica sino en la lógica del negocio o dominio. El problema estaría cuando se intenta resolver problemas del dominio con tecnología. eso difiere que aunque la aplicación esté funcionando correctamente no haya nadie capaz de entender como lo hace.

Por lo cual es muy probable que surja en antipatrón "Modelo del dominio anémico". Pero al hacer DDD evitaremos esta y otras malas prácticas, previniendo que acabemos con una aplicación complicada al momento de mantenerla.

Beneficios de usar técnica para la toma de decisiones del diseño:

- Comunicación efectiva entre expertos del dominio y expertos técnicos..
- Foco en el desarrollo de un área dividida del dominio (subdominio).
- El software es más cercano al dominio, y por lo tanto es más cercano al cliente.
- Código bien organizado, permitiendo el testing de las distintas partes del dominio de manera aisladas.

- Mantenibilidad a largo plazo.

5. REQUISITOS NO FUNCIONALES ELEGIDOS POR GRYFFINDOR

6.1 CERTIFICACIÓN

La certificación se basa en garantizar la satisfacción del usuario mediante el desarrollo de tecnológico , esto se logra mediante pruebas antes de que el producto por así decirlo salga a producción.

Se abarcan diferentes tipos de pruebas como funcionales y no funcionales. con el objetivo de probar los diferentes escenarios que podría tener un usuario final y así localizar los errores y vulnerabilidades que tiene un sistema en un sí vida de desarrollo ya que de lo contrario podrían presentar una mala experiencia para de usuario, perdida de tiempo, pérdida de inversión, etc...

Antes que un desarrollo tecnológico salga a producción debe cumplir con todas las expectativas que espera el cliente y con los estándares de calidad que exigen las normas.

ISO/IEC 25000 además de ser familia de normas para evaluar la calidad de software también certifica desarrollos tecnológicos antes de salir a producción, es muy buena opción para certificar el desarrollo.



6.2 CONFORMIDAD

Al hablar de conformidad como requisito no funcional en el desarrollo de software se refiere a la evaluación independiente de la conformidad de cualquier actividad, proceso, producto final, producto o servicio de los criterios de normas especificadas , mejores prácticas e requisitos documentados.

La conformidad se aplica a todas las características de ISO/IEC 9126, como por ejemplo conformidad a la legislación referente a usabilidad y fiabilidad ISO/IEC 9126, distingue entre fallo y no conformidad. Un fallo es el incumplimiento de los requisitos previos, mientras que la que no conformidad es el incumplimiento de los requisitos especificados.



6.3 GESTIÓN DE CONFIGURACIÓN

Es la encargada de garantizar que todos los activos de software y hardware que posee una empresa sean conocidos y rastreados en todo momento, es decir que se puede pensar a la gestión de la configuración como un inventario siempre actualizado de sus activos tecnológicos, una única fuente de información.

se suelen abarcar en diferentes áreas y se le relaciona con ideas como la creación de “tuberías de software” para construir y probar los artefactos de software a disposición. También está muy relacionado con la escritura de “infraestructura como código” para capturar en código el estado actual de nuestra infraestructura. Con lo que se pueden incorporar herramientas de gestión de configuración para almacenar el estado actual de nuestros servidores.

6.4 TESTABILIDAD

Podemos entender la testabilidad como la facilidad con la que el software permite establecer criterios de prueba, y la ejecución de casos de prueba, de tal manera que se pueda medir luego de ejecutados los casos de prueba, si se han alcanzado esos criterios; La testabilidad hace referencia a todo lo que haga más fácil la realización del testeo de la solución software ya sea porque facilita el diseño de los casos de prueba sobre los cuales se va a evaluar la aplicación, o testear de manera más eficiente.

La capacidad de prueba de los componentes de software (módulos, clases) está determinada por factores tales como:

Controlabilidad: el grado en que es posible controlar el estado del componente bajo prueba (CUT) según sea necesario para la prueba.

Observabilidad: el grado en que es posible observar los resultados de las pruebas (intermedias y finales).

Aislamiento: el grado en que el componente bajo prueba (CUT) se puede probar de forma aislada.

Separación de preocupaciones: el grado en que el componente bajo prueba tiene una responsabilidad única y bien definida.

Comprensibilidad: el grado en que el componente bajo prueba está documentado o se explica por sí mismo.

Automatización: el grado en que es posible automatizar las pruebas del componente bajo prueba.

Heterogeneidad: el grado en que el uso de diversas tecnologías requiere el uso de diversos métodos y herramientas de prueba en paralelo.

6.5 TRANSPARENCIA

La transparencia, como se usa en la ciencia, la ingeniería, los negocios, las humanidades y en otros contextos sociales, está funcionando de tal manera que es fácil para otros ver qué acciones se realizan. La transparencia implica apertura, comunicación y responsabilidad.

La transparencia se practica en empresas, organizaciones, administraciones y comunidades. Por ejemplo, un cajero que realiza un cambio después de una transacción en el punto de venta al ofrecer un registro de los artículos comprados (por ejemplo, un recibo) y al contar el cambio del cliente en el mostrador demuestra un tipo de transparencia.

El término transparencia tiene un significado muy diferente en seguridad de la información, donde se usa para describir mecanismos de seguridad que son intencionalmente inadvertidos u ocultos a la vista. Los ejemplos incluyen herramientas y herramientas ocultas que el usuario no necesita saber para hacer su trabajo, como mantener las operaciones remotas de autenticación del Protocolo de autenticación Challenge-Handshake ocultas para el usuario.

6.6 CAPACIDAD - ACTUAL Y PRONOSTICADA

La capacidad del canal, en ingeniería eléctrica, informática y teoría de la información, es el límite superior estricto en la velocidad a la que la información se puede transmitir de manera confiable a través de un canal de comunicación.

Siguiendo los términos del teorema de codificación de canal ruidoso, la capacidad de canal de un canal dado es la tasa de información más alta (en unidades de información por unidad de tiempo) que se puede lograr con una probabilidad de error arbitrariamente pequeña.

La teoría de la información, desarrollada por Claude E. Shannon en 1948, define la noción de capacidad del canal y proporciona un modelo matemático mediante el cual se puede calcular. El resultado clave establece que la capacidad del canal, como se definió anteriormente, está dada por el máximo de la información mutua entre la entrada y la salida del canal, donde la maximización es con respecto a la distribución de entrada.

La noción de capacidad de canal ha sido fundamental para el desarrollo de sistemas modernos de comunicación por cable e inalámbrica, con el advenimiento de nuevos mecanismos de codificación de corrección de errores que han dado como resultado un rendimiento muy cercano a los límites prometidos por la capacidad del canal.

6.7 DURABILIDAD

La durabilidad del software significa la capacidad de solución de la utilidad del servicio del software y satisfacer las necesidades del usuario durante un tiempo relativamente largo. Va muy relacionado con la “vida útil” de nuestro software, y es un parte a tener muy en cuenta a la hora de adquirir un producto cualquiera.

6.8 DOCUMENTACIÓN

La documentación del software es un texto escrito o ilustración que acompaña al software o está incrustado en el código fuente. La documentación explica cómo funciona el software o cómo usarlo, y puede significar diferentes cosas para las personas en diferentes roles. Esto ayuda a entender las funciones y características principales de un programa más fácilmente.




En teoría todo proceso debe de quedar documentado para que cualquier usuario (programador o no), tenga cierta noción de qué es lo que se trata de hacer en cada parte del código o del proyecto como tal.



6.9 RECUPERACIÓN DE DESASTRES

Se espera tener un plan con herramientas que permitan proporcionar continuidad y recuperación comercial después de desastres naturales o provocados por el hombre y cortes de energía.

Este requisito va mucho más allá de tener un simple backup, ya que no solo garantiza que los datos perdidos se repliquen y almacenen en un lugar seguro para luego restaurarlos, si no que también se espera que se reinstale el software y los datos en máquinas nuevas y luego configurarlos con las preferencias apropiadas.

Existen muchas herramientas hoy en día para realizar esta tarea y así velar porque este requisito no funcional se cumpla, como por ejemplo.

**Veeam Backup & Replication** 
 **★ TOP RATED** 281 Ratings
Veeam Backup & Replication is a backup platform for virtual environments built on VMware vSphere and Microsoft Hyper-V hypervisors. The software provides backup, restoration, and replication functionality for virtual machines, physical servers and workstations as well as cloud-based workloads.
[Reviews \(98\)](#) | [Alternatives](#) | [Download Free Trial](#)

**Carbonite Server Backup**
 57 Ratings
Carbonite (Nasdaq:CARB) is a leading provider of cloud backup and restore solutions for small and midsize businesses. Together with our partners we protect millions of devices and their valuable data for businesses and individuals around the world who rely on us to ensure their important data is ...

REFERENCIAS

<https://www.sfia-online.org/es/sfia-6/skills/procurement-management-support/quality-and-conformance>

<https://www.javiergarzas.com/2015/03/certificar-funcional-aplicacion-software.html>

<https://www.pragma.com.co/academia/conceptos/todo-lo-que-debes-saber-sobre-certificacion-de-software>

https://en.wikipedia.org/wiki/Software_documentation

https://en.wikipedia.org/wiki/Software_documentation

https://en.wikipedia.org/wiki/Software_durability

<https://sdos.es/blog/pruebas-de-rendimiento-con-jmeter-ejemplos-basicos>

<https://www.trustradius.com/disaster-recovery>

<https://www.youtube.com/watch?v=O6xfXrpppFM>

http://cic.puj.edu.co/wiki/lib/exe/fetch.php?media=materias:modelo4_1.pdf