Taller N°1 - Artefactos de diseño del software
Liseth Arelis Giraldo Morales
C.C 1.036.965.346
Ficha 2585840
ADSO
CTGI

1. **Observar la siguiente imagen y mediante un análisis, resolver los siguientes interrogantes:**

   Los artefactos de diseño del software son documentos o entregables que describen los aspectos técnicos y funcionales de un sistema o aplicación. En el contexto de una metodología de desarrollo de software, donde pueden incluir:

   ● **Diagramas de clases:** Representan la estructura de las clases y sus relaciones.
   ● **Diagramas de secuencia:** Muestran la interacción entre los objetos y cómo se realiza una tarea específica.
   ● **Modelos de datos:** Describen la estructura de los datos que se manejan en el sistema.
   ● **Especificaciones técnicas:** Describen detalladamente cómo se construirá el sistema.
   ● **Plan de pruebas:** Describen las pruebas que se realizarán para garantizar la calidad del software.

   Estos son solo algunos ejemplos de artefactos de diseño. La selección de artefactos puede variar dependiendo de la metodología seleccionada. Por ejemplo, en una metodología ágil, los artefactos pueden ser más simples y se enfocan en la colaboración y la comunicación continua con los equipos de desarrollo y los clientes. Mientras que, en una metodología más tradicional, como el proceso Unificado de Desarrollo de Software, puede haber una documentación más detallada y rigurosa.

2. **Realizar el siguiente cuestionario de diseño de artefactos y consignar en el informe semanal de la formación.**

   1. **<u>¿Qué es un caso de uso en el desarrollo de software?</u>**

      Un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico.

      Un caso de uso debe:

      - describir una tarea del negocio que sirva a una meta de negocio
      - tener un nivel apropiado del detalle
      - ser bastante sencillo como que un desarrollador lo elabore en un único lanzamiento

Situaciones que pueden darse:

- Un actor se comunica con un caso de uso (si se trata de un actor primario la comunicación la iniciará el actor, en cambio sí es secundario, el sistema será el que inicie la comunicación).
- Un caso de uso extiende otro caso de uso.
- Un caso de uso utiliza otro caso de uso.

2. ¿Cuál es la importancia de la documentación técnica en el diseño de software?

La documentación técnica adecuada hará que la información sea de fácil acceso y también reducirá la curva de aprendizaje.

Cuando se trata de software, los cambios son casi inevitables. Es posible que su software deba experimentar numerosos cambios debido al entorno empresarial en cambio constante. Una sólida herramienta de documentación de software le permite realizar las modificaciones necesarias en su software sin grandes complicaciones

La documentación, como su propio nombre indica, es todo tipo de información que ayuda a dirigir los esfuerzos del equipo de desarrollo y que además ayude a entender la arquitectura y diseño de la aplicación a lo largo del tiempo. De cara al cliente puede ser toda aquella información que le guíe a la hora usar la aplicación una vez entregada.

La documentación de un programa empieza a la vez que la construcción del mismo. De hecho, lo más normal es comenzar haciendo esquemas de las piezas del programa, funcionalidad, interfaces, etc. para que, en el momento que nos pongamos a programar, tengamos muy claro qué es lo que se va a desarrollar y cómo lo vamos a hacer.

La realización de la documentación no termina con la entrega de la aplicación, pues durante el mantenimiento es necesario actualizarla para reflejar los cambios que se hayan tenido que realizar para crear nuevas funcionalidades.

No confundir documentación con los comentarios del código

Es importante no confundir la documentación con los comentarios del código. En realidad un código debería de ser tan sencillo y claro que no requiera comentarios. Si tienes que comentar un código porque es demasiado complejo como para entenderlo leyendo, es que hay algo que estás haciendo mal.

Por tanto, se deben evitar a toda costa los comentarios al código y, en cambio, tenemos que conseguir programas que leyéndolos podamos entender perfectamente qué es lo que hacen.

3. **¿Cómo se utiliza un diagrama de clases en el desarrollo de software orientado a objetos?**

En ingeniería de software, un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos.

UML proporciona mecanismos para representar los miembros de la clase, como atributos y métodos, así como información adicional sobre ellos.

Para especificar la visibilidad de un miembro de la clase (es decir, cualquier atributo o método), se coloca uno de los siguientes signos delante de ese miembro:

+	Público
-	Privado
#	Protegido
/	Derivado (se puede combinar con otro)
~	Paquete

UML especifica dos tipos de ámbitos para los miembros: instancias y clasificadores y estos últimos se representan con nombres subrayados.

- Los miembros **clasificadores** se denotan comúnmente como "estáticos" en muchos lenguajes de programación. Su ámbito es la propia clase.
    - Los valores de los atributos son los mismos en todas las instancias
    - La invocación de métodos no afecta al estado de las instancias
- Los miembros **instancias** tienen como ámbito una instancia específica.
    - Los valores de los atributos pueden variar entre instancias
    - La invocación de métodos puede afectar al estado de las instancias (es decir, cambiar el valor de sus atributos)

Para indicar que un miembro posee un ámbito de clasificador, hay que subrayar su nombre. De lo contrario, se asume por defecto que tendrá ámbito de instancia.

4. **¿Qué es un modelo de arquitectura de software y cómo se utiliza en el proceso de diseño?**

Una arquitectura de software se entiende como la estructura de un sistema, que contiene componentes, sus propiedades visibles a otros componentes y las relaciones entre ellos.
Forma parte del diseño del sistema a partir de los Requerimientos; es una de las etapas, incluyendo el Diseño Detallado, la Implementación y la Verificación.

El diseño de una arquitectura de software utiliza los conocimientos de programación para planear el diseño general del software de modo que puedan agregarse detalles más adelante, lo cual permite a los equipos de software delimitar el panorama general y comenzar a elaborar un prototipo.

Si los desarrolladores de software siguen los consejos de diseño y las prácticas recomendadas de arquitectura de software, pueden analizar detalladamente las características de su software y decidir cómo diseñar la arquitectura de este.

Como usarlo:

1. Comprende claramente cuáles son tus requisitos
    - Comienza con una visión general
    - Haz un mapa de tus requisitos funcionales
    - Ten en consideración los requisitos no funcionales
2. Comienza a pensar en cada componente
    - Empieza con el "escenario perfecto"
    - Considera y documenta qué implicaciones tienen tus requisitos
    - Espera y realiza el diseño de la arquitectura final más adelante
3. Divide tu arquitectura en "rebanadas"
4. Hacer un prototipo
    - Mantén un historial de cambios riguroso
    - Ten una sola fuente de información
    - Haz diagramas de tus prototipos
5. Identifica y cuantifica los requisitos no funcionales
    - Desempeño
    - Escalabilidad
    - Portabilidad
    - Extensibilidad
    - Cumplimiento normativo

Prácticas recomendadas:

1. Visualiza tu diseño.
2. No elijas patrones.
3. Recuerda que el primer diseño es solo la primera iteración.
4. Ten cuidado con el crecimiento excesivo del alcance.

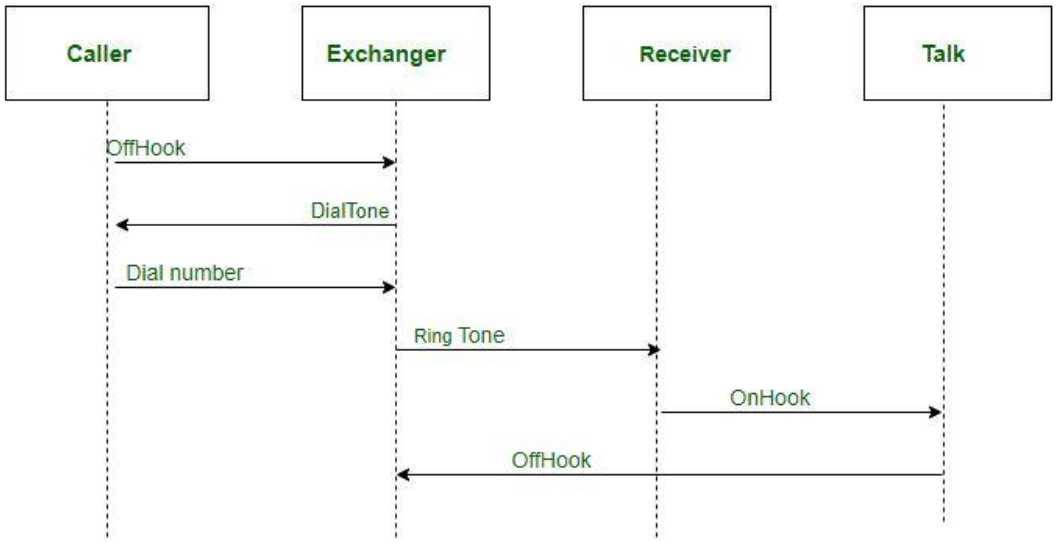
5. **¿Cómo se puede mejorar la calidad del software a través de la revisión de código?**

Una revisión de código es un acto de verificar conscientemente piezas de código en buscade errores y fallas. Después de todo, la codificación es una actividad humana y, por lo tanto, está plagada de errores. Las revisiones de código pueden ejecutarse mediante herramientas/software de revisión de código y humanos. Cuando un desarrollador humano

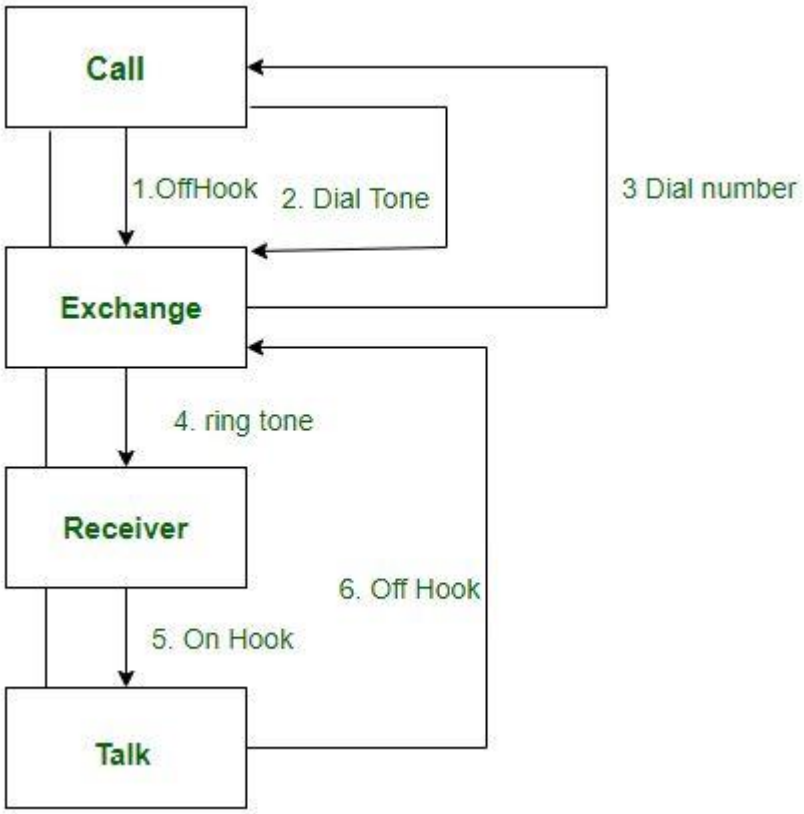
ejecuta la revisión del código, es importante que la persona que verifica y prueba el código no sea la misma persona que escribió el código en primer lugar. Es por eso que a usted, como desarrollador, se le puede pedir que revise los códigos que han escrito sus colegas. Por eso también es importante que sepas cómo funciona el proceso de revisión de código.

6. **¿Cuál es la diferencia entre un diagrama de secuencia y un diagrama de colaboración en UML?**

Un diagrama de secuencia es un diagrama de interacción que detalla la operación que se lleva a cabo. El diagrama de secuencia captura la interacción entre los objetos en el contexto de la colaboración. Los diagramas de secuencia se centran en el tiempo y muestran visualmente el orden de la interacción utilizando el eje vertical del diagrama para representar el tiempo.



El diagrama de colaboración representa la interacción de los objetos para realizar el comportamiento de un caso de uso particular o una parte del caso de uso. Los diseñadores utilizan el diagrama de secuencia y los diagramas de colaboración para definir y aclarar los roles de los objetos que realizan un flujo particular de eventos de un caso de uso.



Diferencias entre diagrama de secuencia y colaboración:

Diagramas de secuencia	Diagramas de colaboración
El diagrama de secuencia representa el UML, que se utiliza para visualizar la secuencia de llamadas en un sistema que se utiliza para realizar una funcionalidad específica.	El diagrama de colaboración también se incluye en la representación UML que se utiliza para visualizar la organización de los objetos y su interacción.
El diagrama de secuencia se utiliza para representar la secuencia de mensajes que fluyen de un objeto a otro.	Los diagramas de colaboración se utilizan para representar la organización estructural del sistema y los mensajes que se envían y reciben.
El diagrama de secuencia se usa cuando la secuencia de tiempo es el enfoque principal.	El diagrama de colaboración se usa cuando la organización de objetos es el enfoque principal.
Los diagramas de secuencia se adaptan mejor a las actividades de análisis.	Los diagramas de colaboración son más adecuados para representar interacciones más simples de la menor cantidad de objetos.

7. **¿Cómo se asegura la escalabilidad en el diseño de software?**

El escalado de aplicaciones se refiere al crecimiento de su aplicación. El crecimiento puede ser en términos de tráfico que la aplicación está recibiendo o evolucionando para satisfacer la necesidad comercial. Como cualquier negocio exitoso depende de la escalabilidad para tener un crecimiento positivo, al igual que la aplicación. En la actualidad, la escalabilidad empresarial va de la mano con el escalado de aplicaciones. Aquí encontrará todo lo que debe saber sobre por qué es importante escalar aplicaciones y cómo puede hacerlo.

¿Qué es la escalabilidad en una aplicación?
Cuando se trata de escalar aplicaciones, generalmente se refiere a la base de datos de la aplicación en crecimiento y sus modificaciones de back-end. Puede haber muchas formas diferentes de escalar la aplicación. Generalmente varía según el tipo de negocio hasta el desarrollo de la aplicación.

¿Por qué su aplicación necesita ser escalable?
Para simplificar, si una aplicación obtiene 100 usuarios por día y su base de usuarios activos se dispara a 1000 usuarios por día. La aplicación con un backend no optimizado no podrá manejar ese tráfico mucho mayor. Ocurre principalmente debido a que no se tuvo en cuenta la escalabilidad durante la fase de desarrollo.

Al igual que un negocio que comienza a tener éxito, se convierte en una necesidad imperiosa escalarlo para entretener a todos sus clientes. Lo mismo ocurre con la aplicación; la escalabilidad se vuelve crucial para sostener a los usuarios y brindar una experiencia agradable al tráfico. Sin embargo, si se pierde la optimización durante la fase de desarrollo de la aplicación, hay soluciones de terceros disponibles que brindan optimización de back-end para la expansión de la aplicación.

8. **¿Qué es un prototipo en el desarrollo de software y para qué se utiliza?**

Un prototipo en sentido genérico es una implementación parcial pero concreta de un sistema o una parte de este que principalmente se crean para explorar cuestiones sobre aspectos muy diversos del sistema durante el desarrollo de este.

En referencia a una interfaz de usuario se realizan prototipos con la finalidad de explorar los aspectos interactivos del sistema incluyendo la usabilidad, la accesibilidad y/o la funcionalidad de este.

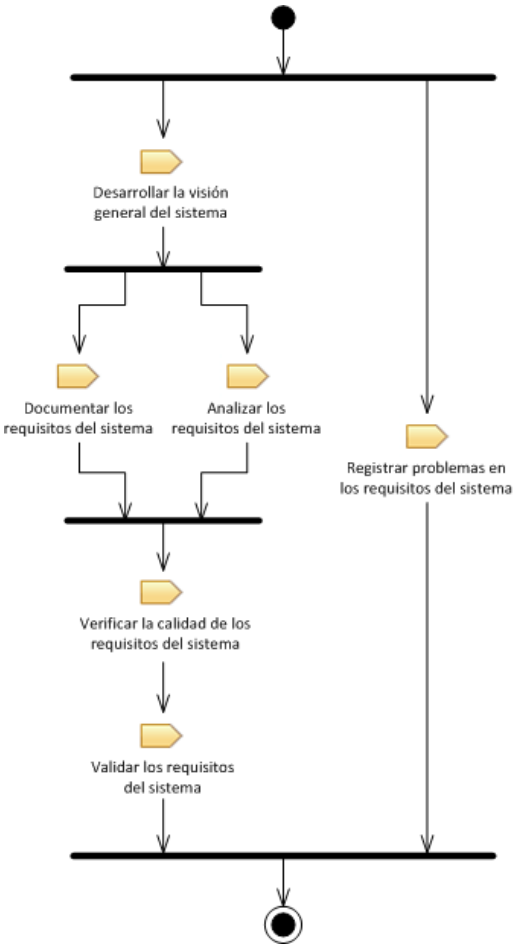
El uso de los prototipos en el desarrollo de sistemas software no se limita sólo a probar las interacciones que los usuarios deben realizar, sino que son útiles también para otras actividades que se realizan durante el proceso, como por ejemplo su gran utilidad en la fase de recogida o análisis de requisitos en cuanto que amplía y mejora y la información necesaria para el desarrollo del sistema

9. ¿Cómo se integran los requisitos del usuario en el proceso de diseño de software?

El objetivo principal de esta actividad es desarrollar una propuesta de sistema software como solución a las necesidades de negocio de clientes y usuarios, haciéndolo con la mayor calidad posible e intentando asegurar que coincide con las expectativas de clientes y usuarios.

Los ingenieros de requisitos deben elaborar la propuesta en forma de requisitos del sistema software a desarrollar (product requirements en terminología CMMI-DEV), usando como entrada la información que se va generando en el procedimiento Identificar las necesidades de negocio de clientes y usuarios. Lo habitual es comenzar por los aspectos más generales del sistema (su visión general), para ir avanzando en el nivel de detalle hasta que se considere suficiente como para que pueda continuarse el desarrollo sin tener que plantear preguntas continuas sobre la conducta del sistema a desarrollar.

Durante esta actividad es frecuente que en cualquier momento se identifiquen y registren diversos tipos de problemas en los requisitos, cuya solución debe gestionarse por la actividad de Gestionar los problemas en los requisitos dentro del procedimiento Gestionar los requisitos del sistema software a desarrollar.



10. **¿Cómo se puede mejorar la eficiencia y el rendimiento en el desarrollo de software?**

Una parte fundamental en todo desarrollo es la dedicada a optimizar software. Aunque habitualmente los recursos de hardware son ya bastante potentes como para ejecutar un software poco optimizado, las posibilidades de este estarán limitadas por un diseño o implementación defectuosos.

No solo eso, también es posible que el software funcione correctamente, pero que haga un uso poco optimizado de los recursos. Esto obliga a aumentar continuamente recursos como la capacidad de disco. Y como consecuencia, aumentar la inversión que hay que hacer en hardware y horas de trabajo. Por tanto, optimizar el software es importante. Eso sí, para ello necesitarás tener algunas pautas y herramientas.

Conviene tener presente que la optimización del software es algo a considerar en todo el proceso de diseño. En algunas ocasiones, parece evidente que un patrón de diseño se adapta bien a una parte de la aplicación. Sin embargo, puede resultar poco eficiente. En algunos casos, no es necesario optimizar algunos componentes al máximo. Esto se debe a que se utilizan poco, o en condiciones en que una pequeña penalización al rendimiento no es crítica.

La cosa cambia cuando se utilizan, por ejemplo, recursos externos muy costosos en tiempo o dinero. Es el caso de multitud de APIs de terceros que, además de requerir esperas por tratarse de recursos a los que se accede por medio de la red, en algunas ocasiones interesa reducir el número de accesos a ellas. Más que nada, porque la facturación por utilizarlas se realiza muchas veces con base en la cantidad de operaciones realizadas. En estos casos, tiene sentido implementar sistemas de caché que aseguren que el acceso solo tiene lugar cuando la información no está almacenada. O bien cuando se ha quedado obsoleta.

Sin embargo, no siempre es posible prever qué componentes del software es necesario optimizar. Prever el comportamiento de algunos componentes o recursos externos es complicado. Además, en ocasiones no es viable curarse en salud y optimizar determinadas funciones solo por si acaso. Hacerlo alargaría los tiempos y costes del desarrollo.

3. **Analizar le infografía y realizar conclusiones de conocimientos que debe tener un FRONTEND**

Todo aquel profesional que sea un Front end Developer deberá tener unas habilidades y conocimientos necesarios.

- Conocer HTML5 y CSS. HyperText Markup Language, es el componente estructural clave de todas las páginas webs que se encuentran en internet. Y el Cascading Style Sheets es lo que le proporciona estilo a HTML.
- Conocimientos en JavaScript. Gracias a JS se podrá conseguir una página web interactiva.
- Saber utilizar un CMS, por ejemplo, podría utilizar WordPress.
- Manejo de librerías y frameworks
- Ser creativo.
- Manejo de repositorios y git
- Tener conocimientos de diseño.
- Conocer el trabajo de un diseñador web y de un desarrollador back-end.
- Conocimiento en despliegues.

4. Solucionar el cuestionario de artefactos de desarrollo de software y consignarlos en el informe semanal de la formación.

1. ¿Qué es un artefacto de desarrollo de software?

   Un artefacto es uno de los muchos tipos de subproductos tangibles producidos durante el desarrollo del software. Algunos artefactos (p. ej., casos de uso , diagramas de clases y otros modelos, requisitos y documentos de diseño del lenguaje de modelado unificado (UML)) ayudan a describir la función, la arquitectura y el diseño del software. Otros artefactos están relacionados con el proceso de desarrollo en sí mismo, como planes de proyectos, casos comerciales y evaluaciones de riesgos.

El término artefacto en relación con el desarrollo de software se asocia en gran medida con métodos o procesos de desarrollo específicos, por ejemplo, Proceso unificado . Este uso del término puede haberse originado con esos métodos [cita requerida] .

En las pruebas no basadas en la ejecución, los artefactos son los recorridos , las inspecciones y las pruebas de corrección . Por otro lado, las pruebas basadas en la ejecución requieren como mínimo dos artefactos: un conjunto de pruebas y el ejecutable. Artefacto ocasionalmente puede referirse al código publicado (en el caso de una biblioteca de código) o ejecutable lanzado (en el caso de un programa) producido, pero más comúnmente un artefacto es el subproducto del desarrollo de software en lugar del producto en sí.

En el desarrollo del usuario final, un artefacto es una aplicación o un objeto de datos complejo que crea un usuario final sin necesidad de conocer un lenguaje de programación general. Los artefactos describen comportamientos automatizados o secuencias de control, como solicitudes de bases de datos o reglas gramaticales, o contenido generado por el usuario .

Los artefactos varían en su mantenibilidad. La mantenibilidad se ve afectada principalmente por el rol que cumple el artefacto. El papel puede ser práctico o simbólico. En las primeras etapas del desarrollo de software, el equipo de diseño puede crear artefactos que desempeñen un papel simbólico para mostrar al patrocinador del proyecto la seriedad con la que el contratista satisface las necesidades del proyecto. Los artefactos simbólicos a menudo transmiten información de manera deficiente, pero tienen un aspecto impresionante.

2. ¿Cuáles son algunos ejemplos comunes de artefactos de desarrollo de software?

Historias de usuario. Las historias de usuario se crean como especificaciones de los requisitos del cliente (Inayat et al., 2015). Éstas facilitan la comunicación y la comprensión general de los requisitos entre los involucrados (Daneva et al., 2013). A diferencia de los métodos tradicionales de especificación de requisitos de software, las historias de usuario cambian el foco de las especificación escrita y completa de los requisitos a una comunicación cara a cara con el cliente (Carlson y Matuzic, 2010). Las historias de usuario enfatizan las metas de usuario, explicando brevemente la percepción del usuario y en enfocándose en que se necesita hacer (Carlson y Matuzic, 2010).

Prototipos. El prototipo es un modelo de la aplicación del software que soporta la comunicación y la evaluación de alternativas (Schön et al., 2017). La creación de prototipo se percibe como una manera directa de revisar la especificación de requisitos con los clientes para obtener una validación oportuna de éstos (Inayat et al., 2015). La construcción del prototipo inicia con los requisitos que han sido seleccionados para abordarse en la siguiente iteración (De Lucia y Qusef, 2010). En particular, aquellos que

se entienden completamente y que tienen alta prioridad. Sin embargo, este enfoque puede ser muy costoso cuando se requiere el desarrollo de múltiples prototipos de alta fidelidad (Inayat et al, 2015).

Casos de uso. Un caso de uso describe un conjunto de secuencia de acciones que ejecuta un sistema para producir un resultado observable, de valor para un actor (Booch, et al., 2006). Un actor representa u rol que es desempeñado por una persona, sistema o dispositivo (Booch et al., 2006 p. 247).

Escenario. Un escenario es una secuencia específicas de acciones que ilustra un comportamiento particular (Booch et al. 2006). Describe textualmente la interacción entre el usuario y el sistema en un contexto particular (Schön et al., 2017).

Tarjeta de historia. La tarjeta de historia es la representación tangible, en texto, de una historia de usuario (Schön et al., 2017). Las tarjetas de historia también pueden capturar estimados, quienes trabajaron en la historia y metas. Otras propuestas de su contenido consideran ocho aspectos: quién, qué, por qué, sin ignorar, mientras sea agradable tenerlo, dentro de, con una prioridad de, y cuyo impacto (Schön et al., 2017).

Persona. Persona es una descripción de una persona ficticia que representa a un conjunto de potenciales usuarios del sistema (Schön et al., 2017).

Visión. La visión es una representación abstracta de la meta general que guía el desarrollo de producto (Schön et al., 2016).

Product backlog. La gestión de requisitos se realiza a través de gestionar el backlog del proyecto (Inayat et al. 2015). El backlog contiene las tarjetas que describen cada historia de usuario.

Tablero Kanban. Para monitorear el flujo de trabajo durante el proyecto de software, el tablero Kanban puede ser utilizado para visualizar el progreso de un requisito.

Respecto de la documentación de requisitos, existen algunos problemas para usarlos durante el proceso de desarrollo En consecuencia, es importante encontrar la combinación apropiada de artefactos según el contexto del proyecto y los miembros del equipo de desarrollo de software (Schön et al., 2017).

3. ¿Cuál es la importancia de los artefactos de desarrollo de software en el proceso de desarrollo de software?

   Los artefactos son los productos que resultan de una actividad (o práctica) y se usan para comunicar, elaborar, validar y especificar los requisitos en ambientes de desarrollo de software. Schön et al. (2017) reportó 57 diferentes artefactos que consideran la participación del cliente en métodos ágiles. Entre ellos se encuentran las historias de

usuario, prototipos, casos de uso, escenarios, tarjeta de historia, Persona, visión, diagramas UML, viñetas (storyboards), tareas, tablero Kanban, patrón de interfaz de usuario, imágenes, videos, mapas mentales, especificación de la interfaz de usuario, entre otros.

4. **¿Qué diferencia hay entre un artefacto y un producto de software?**

   Los Artefactos son herramientas que ayudan a la construcción de un producto, mientras que el producto de software surge de la recolección y construcción de diferentes artefactos, estos dos luego toman parte de un mismo conjunto para llegar a una solución confiable para el cliente o usuario final.

5. **¿Cómo se almacenan y gestionan los artefactos de desarrollo de software en un proyecto de software?**

   La gestión de proyectos de software hace referencia a la rama de la gestión de proyectos dedicada a la planificación, programación, asignación de recursos, ejecución, seguimiento y entrega de proyectos de software y web.

   La gestión de proyectos en ingeniería de software difiere de la gestión de proyectos tradicional en que los proyectos de software tienen un proceso de ciclo de vida único que requiere varias rondas de pruebas, actualizaciones y comentarios de los clientes. La mayoría de los proyectos relacionados con TI se gestionan al estilo Agile, para seguir el ritmo cada vez más rápido del negocio e iterar en función de los comentarios de los clientes y las partes interesadas.

6. **¿Qué papel desempeñan los artefactos de desarrollo de software en la gestión de la configuración de software?**

   Proporcionar una guía de ejecución del proyecto que defina para los técnicos la secuencia de tareas que se requieren y los productos que deben generar.
   Mejorar la calidad del producto en:
   - Disminuir el número de fallos
   - Bajar la severidad de los defectos
   - Mejorar la reusabilidad
   - Mejorar la estabilidad del desarrollo y el costo de mantenibilidad

   Mejorar la predecibilidad del proyecto en:
   - La cantidad de trabajo que requiere
   - El tiempo de desarrollo que se necesita

   Generar la información adecuada a los diferentes responsables de forma que ellos puedan hacer un seguimiento efectivo.

7. ¿Cómo se utilizan los artefactos de desarrollo de software en la integración continua y el despliegue continuo?

El proceso de desarrollo de software está compuesto por una serie de procesos donde participan diferentes departamentos que actúan en distintos entornos. Para poder optimizar todo el ciclo de vida del software y conseguir un producto de mayor calidad reduciendo posibles errores, metodologías como DevOps apuestan por la automatización en las etapas de desarrollo, así como una mejor comunicación entre los desarrolladores y profesionales operadores de TI.

Para alcanzar estos objetivos de un software de mayor calidad, un mejor flujo de trabajo y una reducción de costes y errores, se utilizan técnicas como integración continua, entrega continua o despliegue continuo.

Integración continua

La integración continua CI/CD (continuous integration) es una práctica por la cual los desarrolladores integran o combinan el código en un repositorio común, facilitando la realización de test o pruebas para detectar y resolver posibles errores. Con la CI se impide que se desarrollen distintas divisiones de una aplicación que luego puedan tener conflictos entre sí.

Los desarrolladores integran periódicamente su código en el repositorio central (como GitHub) en lugar de realizarlo de forma aislada al final del ciclo de producción (como se realizaba de forma tradicional). De esta forma se descubren antes los conflictos entre los nuevos códigos y los existentes, haciendo que la resolución de los mismos sea más sencilla y acarree un menor coste.

En la integración continua, los test son fundamentales para poder garantizar que el código es confiable y será necesario contar con un sistema automatizado para poder realizar todo el proceso de integración de código y test (pruebas unitarias o testing de resolución, por ejemplo).

En el desarrollo moderno de aplicaciones, los equipos están compuestos por distintos desarrolladores que trabajan de forma simultánea en distintas funciones del programa. Con la integración continua estos desarrolladores podrán ir integrando o probando sus cambios y comprobando que funcionen de forma correcta y no entran en conflicto con los cambios de otros desarrolladores.

En el desarrollo de software, la integración continua es uno de los pilares en los que se basan metodologías como DevOps para conseguir reducir costes a la vez que se incrementa la calidad y agilidad en el desarrollo.

Entrega continua

La entrega continua CI/CD (continuous delivery) está relacionada con la integración continua y consiste en la automatización del proceso de entrega del software, permitiendo que pueda ser implementado en producción de forma confiable y sencilla. De forma práctica se puede entender la CD como la entrega de actualizaciones de software a los usuarios o clientes de forma sólida y continua.

La entrega continua en DevOps automatiza todo este proceso, desde que se integra el código en el repositorio único, se realizan las pruebas y comprueban los cambios, hasta la entrega al usuario.

La entrega continua se la conoce como el siguiente paso o extensión de la integración continua. Tiene como objetivo principal conseguir que la entrega de los nuevos cambios o actualizaciones del software a los clientes se realicen de forma ágil y fiable.

A pesar de que la automatización es una de las características de la entrega continua, esta fase del desarrollo del software tiene un componente de intervención humana (las entregas se realizarán cuando se indique de forma explícita). En la entrega continua los desarrolladores controlan cuándo y cuántas veces se realiza la entrega de software.

La entrega continua acelera la entrega de software a los usuarios y puede ser realizada varias veces al día, a la semana, o según las necesidades o características de cada proyecto de desarrollo.

Despliegue continuo

Una de las mayores confusiones a la hora de definir estos términos relacionados con el desarrollo de software, se produce por la coincidencia de las siglas de entrega continua y despliegue continuo (en ambas se usa CD).

El despliegue continuo o CD (continuous deployment) está relacionado estrechamente con la entrega continua. Con el despliegue continuo se va un paso más allá de la entrega continua, automatizando todo el proceso de entrega de software al usuario, eliminando la acción manual o intervención humana necesaria en la entrega continua.

El despliegue continuo permite cumplir con el concepto de producción continua que persigue la metodología DevOps. Todo el proceso de despliegue sigue una serie de pasos que deben ejecutarse en orden y de forma correcta. Si alguno de estos pasos no se concluye de forma satisfactoria, el despliegue no se llevará a cabo. Por eso es

fundamental que el diseño de la automatización de pruebas se realice de forma correcta, pues al no producirse ninguna entrada o acción manual, dependerá en gran medida de cómo sea ese diseño.

El despliegue continuo libera de carga a los equipos de operaciones de procesos manuales, que son una de la principal causa de retrasos en la distribución de aplicaciones.

El uso de estas prácticas de mejora en los procesos de desarrollo de software aporta una serie de beneficios como la entrega o liberación inmediata de código, una simplificación del trabajo colaborativo o de equipo, permite una detección temprana de errores, disminuye costes de desarrollo (ahorra tiempo y esfuerzo) y favorece la comunicación entre todos los implicados en el proceso de desarrollo.

Las necesidades actuales a la hora de desarrollar y entregar software de calidad a los clientes de forma periódica, ha propiciado que metodologías como DevOps hayan sido adoptadas en la industria del desarrollo de software para poder automatizar procesos y aplicar técnicas como la integración continua, la entrega continua y el despliegue continuo. Estas técnicas ayudan a que los desarrolladores puedan entregar su código de forma periódica, compartiéndolo en un repositorio común donde puede ser comprobado y corregido, para así poder entregar el software al cliente de forma rápida y fiable.

Implementar estas técnicas es un proceso que puede resultar complejo y que requiere de un cambio en la filosofía y en la manera de hacer las cosas de las empresas de desarrollo de software. Los resultados serán un software de mayor calidad libre de errores que podrá ser entregado al cliente de forma más rápida y periódica.

8. ¿Cómo se asegura la calidad de los artefactos de desarrollo de software?

Para llevar a cabo el aseguramiento de la calidad del software de manera adecuada, deben recabarse, evaluarse y divulgase datos sobre el proceso de ingeniería de software. Los métodos estadísticos aplicados al ACS ayudan a mejorar la calidad del producto y del proceso de software mismo.

El aseguramiento de la calidad del software es un área importante de proceso, que persigue evaluar la calidad del producto de software, en gran medida, mediante el establecimiento de un plan de aseguramiento de la calidad.

Este plan de aseguramiento de la calidad, dentro de su marco de revisión formal, establece llevar a cabo técnicas de evaluación estática de los componentes del software, y técnicas de evaluación dinámicas, las cuales consisten en una secuencia de pruebas

que miden cada escenario potencial donde existe un mínimo riesgo de que el producto no cumpla las expectativas del cliente.

Los individuos altamente capacitados de un equipo de desarrollo muchas veces tienen suficiente dominio, documentación, y además cuentan con las herramientas correctas para realizar las inspecciones del código fuente, hacer las pruebas de usabilidad, las pruebas unitarias, las pruebas de integración, las pruebas de aceptación, las pruebas de regresión, entre otras.

9. ¿Cómo se asegura la trazabilidad de los artefactos de desarrollo de software a lo largo del ciclo de vida del software?

El ciclo de vida del desarrollo del software (también conocido como SDLC o Systems Development Life Cycle) contempla las fases necesarias para validar el desarrollo del software y así garantizar que este cumpla los requisitos para la aplicación y verificación de los procedimientos de desarrollo, asegurándose de que los métodos usados son apropiados.

Su origen radica en que es muy costoso rectificar los posibles errores que se detectan tarde en la fase de implementación. Utilizando metodologías apropiadas, se podría detectar a tiempo para que los programadores puedan centrarse en la calidad del software, cumpliendo los plazos y los costes asociados.

La metodología para el desarrollo de software es un modo sistemático de realizar, gestionar y administrar un proyecto para llevarlo a cabo con grandes posibilidades de éxito. Esta sistematización indica cómo se divide un proyecto en módulos más pequeños para normalizar cómo se administra el mismo.

Así, una metodología para el desarrollo de software son los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado.

De esta forma, las etapas del desarrollo de software son las siguientes:

- Planificación
- Análisis
- Diseño
- Implementación
- Pruebas
- Instalación o despliegue
- Uso y mantenimiento

Es un largo camino desde los requisitos iniciales hasta los resultados finales, y hay muchas cosas que pueden salir mal en el camino. Para asegurar que los entregables mantengan la alineación con los requisitos del negocio, los gerentes de proyecto deben identificar, rastrear y rastrear los requisitos desde sus orígenes, a través de su desarrollo y especificación, hasta su posterior implementación y uso, aprovechando el poder de una Matriz de Trazabilidad de Requisitos (RTM) y Software de gestión de requisitos (RM).

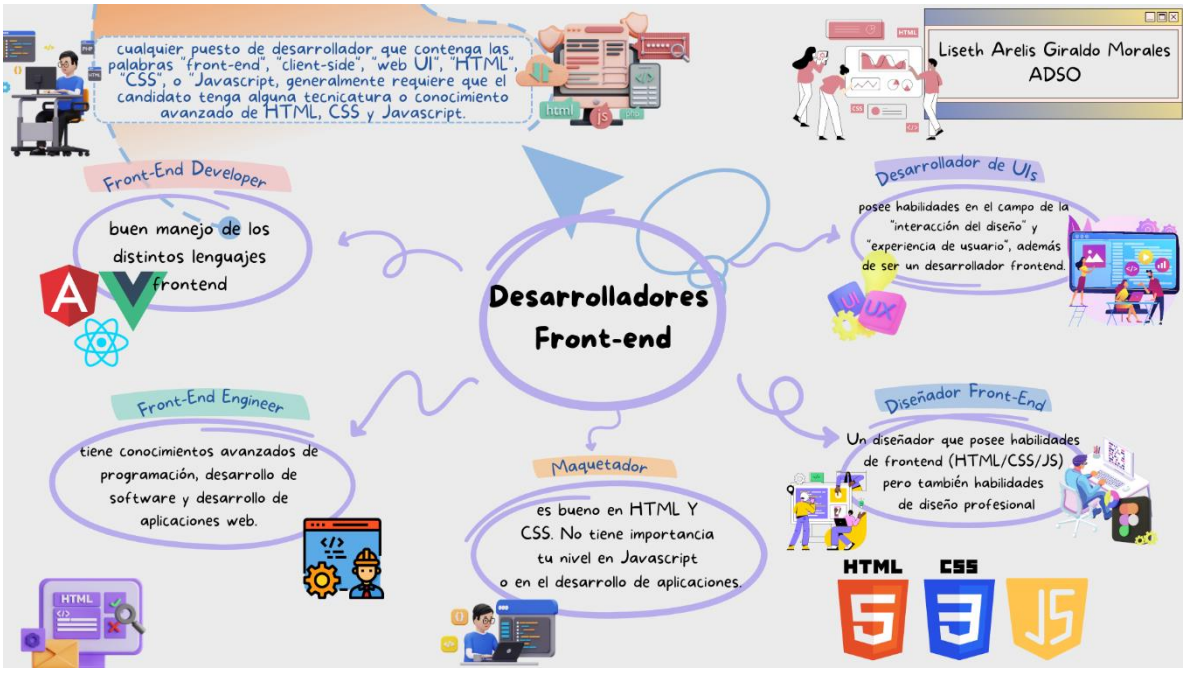
cuando la mayoría de la gente dice “trazabilidad”, lo que realmente quiere decir es trazabilidad de los requisitos, que se define como la capacidad de describir y seguir la vida de un requisito tanto en una dirección hacia delante como hacia atrás en el ciclo de vida del desarrollo, desde sus orígenes hasta su despliegue y más allá.

El objetivo de la trazabilidad de los requisitos es proporcionar visibilidad sobre los requisitos y permitir verificar fácilmente si se cumplen los requisitos. La trazabilidad de los requisitos también ayuda a analizar el impacto de los cambios al revelar cómo un cambio realizado en un requisito afecta a otros requisitos.

Los requisitos pueden ser rastreados manualmente o utilizando varias herramientas de software de rastreo de requisitos. Las Herramientas de software de seguimiento de requisitos hacen que el proceso sea mucho menos engorroso y propenso a errores, y vienen con una serie de características adicionales para proporcionar una forma sistemática de documentar, analizar y priorizar los requisitos.

5. realizar mapa mental de los frontend

https://www.trabajofrontend.com/blog/5-tipos-desarrolladores-frontend
Información analítica del link entregado en esta guía



6. **Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje**

   Dentro de la competencia y el enfoque queda el resultado de aprendizaje se estará trabajando artefactos de desarrollo de software, según la línea de profundización del centro, para ello los temas a contextualización son:

   Los artefactos de diseño del software son documentos que describen y describen cómo se desarrollará un software en términos de su funcionalidad, estructura y comportamiento. En el caso de HTML, CSS y JavaScript, algunos de los artefactos de diseño más comunes incluyen:

   1. Wireframes: Es una representación básica y simplificada de cómo se verá una página web. Se utiliza para definir la estructura de la página y cómo se distribuirán los elementos en la pantalla.
   2. Maquetas: Son versiones más detalladas de los wireframes, que incluyen más información visual y estética sobre cómo se verá una página web.
   3. Especificaciones técnicas: Describen los requisitos técnicos específicos para el desarrollo de una aplicación web, incluyendo detalles sobre la funcionalidad, la estructura de la página y la interacción del usuario con la página.
   4. Diagramas de flujo: Muestran cómo se desarrollarán las interacciones y los procesos en una aplicación web, incluyendo cualquier navegación, procesamiento de datos y eventos de usuario.
   5. Uso de plantillas y mejoras de ellas para páginas web.
   6. Pruebas de usuario: Describen cómo se probarán las funcionalidades y se evaluará la experiencia del usuario en la aplicación web.
   7. El desarrollo de la parte frontal (front-end) de una aplicación web implica conocimientos en varias áreas, tales como:

   - **HTML:** Lenguaje de marcado para crear estructuras y contenido en una página web.
   - **CSS:** Lenguaje de estilo para dar formato y diseño a las páginas web.
   - **JavaScript:** Lenguaje de programación que permite añadir interactividad y dinamismo a las páginas web.
   - **DOM (Document Object Model):** Modelo de objetos que representa el contenido y estructura de una página web.

BIBLIOGRAFÍA

- https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416#:~:text=En%20ingenier%C3%ADa%20del%20software%2C%20un,o%20una%20actualizaci%C3%B3n%20de%20software
- https://desarrolloweb.com/articulos/importancia-documentacion.html
- https://es.wikipedia.org/wiki/Diagrama_de_clases
- https://www.lucidchart.com/blog/es/como-disenar-una-arquitectura-de-software
- https://appmaster.io/es/blog/revisiones-de-codigo
- https://barcelonageeks.com/diferencia-entre-diagrama-de-secuencia-y-diagrama-de-colaboracion/
- https://appmaster.io/es/blog/la-escalabilidad-es-importante
- https://mpiua.invid.udl.cat/fases-mpiua/prototipado/que-es-un-prototipo/#:~:text=Un%20prototipo%20en%20sentido%20gen%C3%A9rico,durante%20el%20desarrollo%20del%20mismo
- https://www.juntadeandalucia.es/servicios/madeja/contenido/procedimiento/20
- https://go4it.solutions/es/blog/optimizar-software-por-que-es-importante
- https://www.iebschool.com/blog/funciones-front-end-developer-analitica-usabilidad/
- https://hmn.wiki/es/Artifact_(software_development)
- https://www.wrike.com/es/project-management-guide/faq/que-es-la-gestion-de-proyectos-de-software/
- https://www.unibe.edu.do/aseguramiento-de-la-calidad-del-software-impulsada-por-metodologias-agiles/
- https://visuresolutions.com/es/what-is-traceability/?amp=1
- https://intelequia.com/blog/post/2083/ciclo-de-vida-del-software-todo-lo-que-necesitas-saber