PD- 2

1.- Reflexiona sobre 3 habilidades técnicas asociadas a la fase de Requerimientos. Indica su importancia dentro del ciclo de Ingeniería de Requerimientos. Incluye argumentos de tu selección. Proporciona un ejemplo basado en la experiencia del proyecto en equipo que están desarrollando.

Empatía:
la capacidad de ponerse en el lugar del usuario y entender sus necesidades y expectativas. Esta habilidad es fundamental para definir los requisitos que realmente satisfagan a los usuarios.

Comunicación:
la capacidad de transmitir información de manera clara, concisa y efectiva. Esta habilidad es necesaria para recopilar los requisitos de los usuarios y comunicarlos al equipo de desarrollo.

Organización:
la capacidad de planificar y gestionar el proceso de recopilación y análisis de requisitos. Esta habilidad es importante para garantizar que el proceso se realice de manera eficiente y eficaz.

La importancia de estas habilidades dentro del ciclo de Ingeniería de Requerimientos radica en que son esenciales para la definición de los requisitos que deben cumplir los sistemas de software. Los requisitos deben ser completos, precisos, consistentes y verificables. Para ello, es necesario que los ingenieros de requisitos sean capaces de recopilar información de los usuarios, analizarla y organizarla de manera adecuada.

En el caso de la plataforma web OptiEstudio, estas habilidades fueron fundamentales para el éxito del proyecto. El equipo de desarrollo utilizó estas habilidades para comprender las necesidades y expectativas de los usuarios, recopilar y analizar los requisitos de manera eficiente y eficaz, y documentar los requisitos de una manera clara y concisa.

Ejemplo de habilidad técnica: Comunicación

Aplicación en el proyecto OptiEstudio:

El equipo de desarrollo de OptiEstudio utilizó una variedad de técnicas de comunicación para recopilar y analizar los requisitos de los usuarios. Estas técnicas incluyeron entrevistas, encuestas, grupos de discusión y prototipos.
Un ejemplo específico de cómo se utilizó la comunicación en el proyecto fue la realización de entrevistas con estudiantes y profesores de diferentes niveles académicos. Estas entrevistas revelaron que los usuarios buscaban una plataforma que les ayudara a mejorar su concentración, comprensión y retención de la información.
El equipo de desarrollo utilizó esta información para diseñar una plataforma que incluyera una variedad de herramientas y recursos diseñados para satisfacer las necesidades de los usuarios. Por ejemplo, la plataforma incluye una función de "entorno de estudio" que permite a los usuarios crear un entorno personalizado con sonidos ambientales y música.

La comunicación también fue importante para el equipo de desarrollo a la hora de documentar los requisitos de la plataforma. El equipo de desarrollo utilizó un lenguaje claro y conciso para documentar los requisitos de una manera que fuera fácil de entender por los desarrolladores y otros miembros del equipo.

Argumento:

La comunicación es una habilidad técnica esencial para los ingenieros de requisitos. Los ingenieros de requisitos deben ser capaces de comunicarse de manera efectiva con los usuarios, los desarrolladores y otros miembros del equipo. Esto es necesario para recopilar y analizar los requisitos de manera precisa y completa.

En el caso de OptiEstudio, la comunicación fue fundamental para que el equipo de desarrollo entendiera las necesidades y expectativas de los usuarios. La información recopilada a través de la comunicación se utilizó para diseñar una plataforma que satisface las necesidades de los usuarios y que está ayudando a mejorar el desempeño académico y la concentración de los estudiantes.

2.- Explica de forma clara la diferencia entre Especificación de Casos de Uso vs Historias de Usuario. Adicionalmente indica en qué casos se recomiendan el uso de cada una de estas herramientas.
Proporciona un ejemplo basado en la experiencia del proyecto en equipo que están desarrollando

Especificación de casos de uso

Una especificación de casos de uso es una descripción formal de un conjunto de interacciones entre un sistema y sus usuarios. Una especificación de casos de uso típica incluye los siguientes elementos:

- **Actores:**

Los actores son los usuarios o agentes externos que interactúan con el sistema.
Casos de uso: Los casos de uso son los escenarios de interacción entre los actores y el sistema.
Precondiciones: Las precondiciones son las condiciones que deben cumplirse antes de que se pueda ejecutar un caso de uso.

- **Postcondiciones:**

Las postcondiciones son las condiciones que se cumplen después de que se ejecuta un caso de uso.
Flujo principal: El flujo principal es la secuencia de pasos que se siguen para completar un caso de uso.

- **Flujo alternativo:**

Los flujos alternativos son las secuencias de pasos que se siguen cuando se produce una condición excepcional.

Historias de usuario

Una historia de usuario es una descripción informal de un requisito desde la perspectiva del usuario. Una historia de usuario típica incluye los siguientes elementos:

- **Nombre:** El nombre es un resumen del requisito.
- **Actor:** El actor es el usuario o agente externo que interactúa con el sistema.
- **Breve descripción:** Una breve descripción del requisito.
- **Precondiciones:** Las precondiciones son las condiciones que deben cumplirse antes de que se pueda completar la historia de usuario.

- **Postcondiciones:**

Las postcondiciones son las condiciones que se cumplen después de que se completa la historia de usuario.

- **Aceptación:**

Los criterios de aceptación son los criterios que se utilizan para determinar si una historia de usuario se ha completado correctamente.

Diferencias entre Especificación de Casos de Uso vs Historias de Usuario

La principal diferencia entre las especificaciones de casos de uso y las historias de usuario es que las especificaciones de casos de uso son más formales y detalladas, mientras que las historias de usuario son más informales y concisas.

Otras diferencias entre estas dos técnicas incluyen:

- **Enfoque:** Las especificaciones de casos de uso se centran en los requisitos funcionales, mientras que las historias de usuario se centran en los requisitos de valor.
- **Propósito:** Las especificaciones de casos de uso se utilizan para documentar los requisitos de un sistema, mientras que las historias de usuario se utilizan para priorizar y comunicar los requisitos de un sistema.
- **Uso:** Las especificaciones de casos de uso se utilizan en proyectos de desarrollo de software de todos los tamaños, mientras que las historias de usuario se utilizan con mayor frecuencia en proyectos ágiles.

Casos de uso

Crear un entorno de estudio personalizado
Actor: Estudiante
Precondiciones: El estudiante tiene una cuenta en OptiEstudio.
Flujo principal:

1. El estudiante abre la plataforma OptiEstudio.
2. El estudiante selecciona la opción "Crear entorno de estudio".
3. El estudiante selecciona el tipo de música, los sonidos ambientales y el color del fondo que desea para su entorno de estudio.
4. El estudiante guarda los cambios.

Postcondiciones: El estudiante tiene un entorno de estudio personalizado que se adapta a sus preferencias individuales.

Historias de usuario

Crear un entorno de estudio personalizado
Actor: Estudiante
Breve descripción: El estudiante desea crear un entorno de estudio personalizado que se adapte a sus preferencias individuales.
Precondiciones: El estudiante debe tener una cuenta en OptiEstudio.
Postcondiciones: El estudiante debe poder personalizar el entorno de estudio según sus preferencias individuales, como el tipo de música, los sonidos ambientales y el color del fondo.
Aceptación: El estudiante debe estar satisfecho con el entorno de estudio personalizado.

Los casos de uso y las historias de usuario presentados son ejemplos de cómo se podrían utilizar estas técnicas para documentar los requisitos de la plataforma OptiEstudio.
Los casos de uso son una buena opción para describir los requisitos funcionales de un sistema. En este caso, los casos de uso "Crear un entorno de estudio personalizado" y "Acceder a una amplia gama de recursos de aprendizaje" describen las funcionalidades principales de la plataforma OptiEstudio.

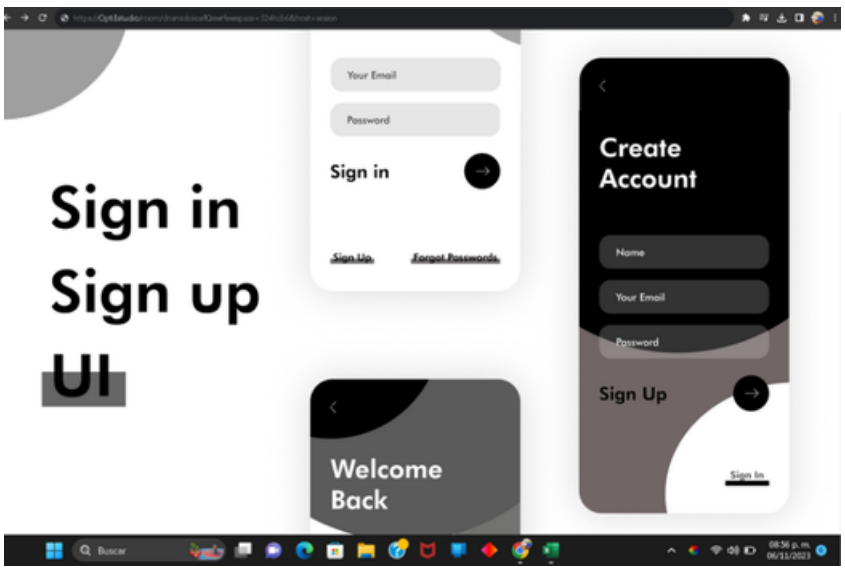
Las historias de usuario son una buena opción para describir los requisitos de valor de un sistema. En este caso, las historias de usuario "Crear un entorno de estudio personalizado" y "Acceder a una amplia gama de recursos de aprendizaje" describen cómo los usuarios interactúan con la plataforma y cómo se benefician de ella.
El equipo de desarrollo de OptiEstudio podría utilizar estos ejemplos como punto de partida para crear un conjunto completo de requisitos para la plataforma. El equipo podría trabajar con usuarios para refinar y priorizar estos requisitos para garantizar que satisfagan las necesidades de los usuarios.

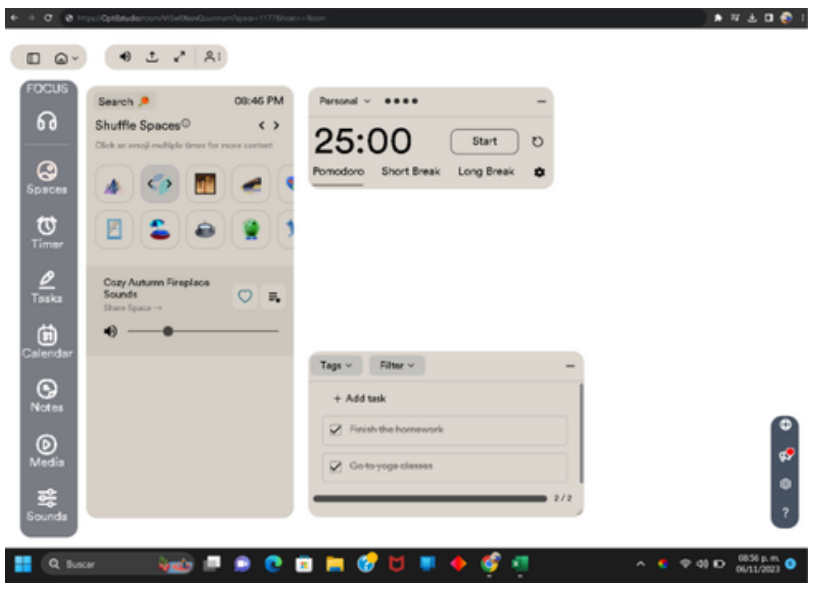
Selecciona un producto resultante de la etapa de diseño (Arquitectura, Base de Datos, Interfaz de Usuario, Procedimientos). Explica de forma clara y sintética algún método asociado al producto resultante. Proporciona un ejemplo basado en la experiencia del proyecto en equipo que están desarrollando.

Selecciona un producto resultante de la etapa de diseño (Arquitectura, Base de Datos, Interfaz de
Usuario, Procedimientos). Explica de forma clara y sintética algún método asociado al producto resultante. Proporciona un ejemplo basado en la experiencia del proyecto en equipo que están desarrollando.

En el proyecto OptiEstudio, el equipo de desarrollo utilizó el prototipado para diseñar la interfaz de usuario de la plataforma. El prototipado es un método que se utiliza para crear versiones preliminares de un producto para su evaluación y mejora.
En el caso de OptiEstudio, el equipo de desarrollo utilizó el prototipado de papel para crear versiones iniciales de la interfaz de usuario. Estas versiones iniciales se utilizaron para obtener comentarios de los usuarios y refinar el diseño de la interfaz.
Una vez que el equipo de desarrollo obtuvo comentarios positivos de los usuarios sobre las versiones iniciales del prototipado, se pasó a crear prototipos de alta fidelidad utilizando herramientas de diseño como Figma o Adobe XD. Estos prototipos se utilizaron para probar la funcionalidad de la interfaz de usuario y garantizar que fuera fácil de usar.
Análisis:
El prototipado es una herramienta valiosa que se puede utilizar para diseñar interfaces de usuario efectivas. El prototipado permite a los equipos de desarrollo obtener comentarios de los usuarios temprano en el proceso de diseño, lo que ayuda a garantizar que la interfaz de usuario satisfaga las necesidades de los usuarios.
En el caso de OptiEstudio, el prototipado resultó ser una herramienta eficaz para diseñar una interfaz de usuario que fuera fácil de usar y que satisficiera las necesidades de los usuarios.
Otros métodos asociados:

Además del prototipado, otros métodos asociados al producto resultante de la etapa de diseño de la interfaz de usuario incluyen:
Diseño centrado en el usuario: Este enfoque de diseño se basa en la comprensión de las necesidades y deseos de los usuarios.
Ergonomía: Este campo de estudio se centra en el diseño de productos que sean fáciles de usar y seguros.
Usabilidad: Este campo de estudio se centra en la facilidad de uso de los productos.





Investiga sobre un método o técnica que permita incluir pruebas de forma efectiva en el proceso de
desarrollo. ¿A partir de qué etapa pueden ser consideradas las pruebas?. Proporciona un ejemplo basado en la experiencia del proyecto en equipo que están desarrollando.

Método o técnica: Ingeniería de pruebas

La ingeniería de pruebas es un enfoque sistemático para la planificación, diseño, implementación, ejecución y evaluación de pruebas de software. La ingeniería de pruebas se puede utilizar para probar cualquier tipo de software, desde aplicaciones web y móviles hasta sistemas integrados y software de misión crítica.

Etapas en las que pueden considerarse las pruebas:

Las pruebas se pueden considerar en todas las etapas del proceso de desarrollo de software, desde la etapa de requisitos hasta la etapa de implementación.

Etapa de requisitos:
En esta etapa, las pruebas se pueden utilizar para validar los requisitos del software. Esto se puede hacer mediante la realización de pruebas de aceptación del usuario (UAT), que son pruebas realizadas por usuarios reales para garantizar que el software satisfaga sus necesidades.

Etapa de diseño:
En esta etapa, las pruebas se pueden utilizar para validar el diseño del software. Esto se puede hacer mediante la realización de pruebas de diseño, que son pruebas que se centran en la estructura y el comportamiento del software.
Etapa de implementación: En esta etapa, las pruebas se pueden utilizar para validar la implementación del software. Esto se puede hacer mediante la realización de pruebas de unidad, pruebas de integración y pruebas de sistema.

Ejemplo basado en la experiencia del proyecto en equipo que están desarrollando:

En el proyecto OptiEstudio, el equipo de desarrollo utilizó la ingeniería de pruebas para probar todas las etapas del proceso de desarrollo.

En la etapa de requisitos, el equipo realizó UAT para validar los requisitos del software con los usuarios. En la etapa de diseño, el equipo realizó pruebas de diseño para validar la estructura y el comportamiento del software. En la etapa de implementación, el equipo realizó pruebas de unidad, pruebas de integración y pruebas de sistema para validar la implementación del software.

El uso de la ingeniería de pruebas en todas las etapas del proceso de desarrollo ayudó al equipo a garantizar que el software de OptiEstudio fuera de alta calidad y que satisficiera las necesidades de los usuarios.

Otros métodos o técnicas:

Otros métodos o técnicas que permiten incluir pruebas de forma efectiva en el proceso de desarrollo incluyen:

Pruebas unitarias: Son pruebas que se centran en la funcionalidad individual de las unidades de código.
Pruebas de integración: Son pruebas que se centran en la interacción entre las unidades de código.
Pruebas de sistema: Son pruebas que se centran en el comportamiento del sistema como un todo.
Pruebas de aceptación del usuario: Son pruebas realizadas por usuarios reales para garantizar que el software satisfaga sus necesidades.

Utilizando fuentes confiables, establece las habilidades/conocimientos/competencias míninas indispensables que un Ingeniero de Software debe incluir en su formación académica para el desarrollo de Aplicaciones de Software Seguras.

Según el Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST), las habilidades/conocimientos/competencias mínimas indispensables que un Ingeniero de Software debe incluir en su formación académica para el desarrollo de Aplicaciones de Software Seguras son las siguientes:

- Conocimientos básicos de seguridad informática: Esto incluye comprender los conceptos básicos de la seguridad informática, como los tipos de amenazas y vulnerabilidades, los métodos de ataque y las medidas de protección.
- Conocimientos de ingeniería de seguridad: Esto incluye comprender los principios de la ingeniería de seguridad, como la evaluación de riesgos, la gestión de la seguridad y el desarrollo de software seguro.
- Conocimientos de lenguajes de programación y tecnologías: Esto incluye comprender los conceptos básicos de los lenguajes de programación y las tecnologías utilizadas en el desarrollo de software, como los lenguajes de programación seguros, las técnicas de desarrollo seguro y las herramientas de seguridad.

Además de estos conocimientos, los Ingenieros de Software también deben desarrollar las siguientes habilidades y competencias:

- **Capacidad de análisis y resolución de problemas:** Esto es esencial para identificar y mitigar las vulnerabilidades de seguridad.
- **Capacidad de pensamiento crítico:** Esto es esencial para evaluar los riesgos de seguridad y tomar decisiones informadas.
- **Capacidad de comunicación efectiva:** Esto es esencial para colaborar con otros profesionales de la seguridad y comunicar los riesgos de seguridad a los usuarios.

La formación académica de los Ingenieros de Software debe incluir cursos y módulos que aborden estos temas. Los programas educativos que ofrecen una formación específica en seguridad informática son una buena opción para los estudiantes que desean desarrollar una carrera en el desarrollo de aplicaciones de software seguras.

Ejemplos de cursos y módulos que pueden incluirse en la formación académica de los Ingenieros de Software para el desarrollo de aplicaciones de software seguras:

- Introducción a la seguridad informática
- Ingeniería de seguridad
- Criptografía
- Seguridad de aplicaciones web
- Seguridad de sistemas operativos
- Seguridad de redes
- Seguridad de la información

Los Ingenieros de Software también pueden adquirir conocimientos y habilidades adicionales en seguridad informática a través de la formación continua y la participación en actividades profesionales, como conferencias, talleres y grupos de trabajo.

PD- 3

Asumiendo que los valores descritos en el acuerdo del "Manifiesto para el desarrollo ágil de software", causan conflicto en el proceso de desarrollo reflexione y describa de manera clara posibles soluciones utilizando como base el proyecto que están desarrollando en equipo.

Los valores descritos en el Manifiesto para el Desarrollo Ágil de Software (Agile Manifesto) son:

Individuales e interacciones sobre procesos y herramientas Software funcionando sobre documentación exhaustiva
Colaboración con el cliente sobre negociación contractual Responder a cambios sobre seguir un plan
Ingeniería de software centrada en las personas sobre herramientas y procesos
Estos valores son fundamentales para el desarrollo ágil de software, ya que promueven la colaboración, la comunicación y la flexibilidad. Sin embargo, también pueden causar conflicto en el proceso de desarrollo, especialmente cuando los equipos no están familiarizados con ellos o no están dispuestos a comprometerse con ellos.

En el proyecto OptiEstudio, los valores del Manifiesto Ágil podrían causar conflicto en las siguientes áreas:

Individuos e interacciones sobre procesos y herramientas: El equipo de desarrollo podría centrarse en las interacciones entre los miembros del equipo y los usuarios, en lugar de seguir estrictamente los procesos y herramientas establecidos. Esto podría conducir a retrasos en el desarrollo o a la entrega de un producto que no cumple con los requisitos.
Software funcionando sobre documentación exhaustiva: El equipo de desarrollo podría centrarse en la entrega de software funcional, en lugar de crear documentación exhaustiva. Esto podría dificultar la comprensión y la mantenimiento del software.

Colaboración con el cliente sobre negociación contractual: El equipo de desarrollo podría colaborar estrechamente con el cliente para comprender sus necesidades y expectativas, en lugar de negociar un contrato que establezca claramente los requisitos y los compromisos. Esto podría conducir a malentendidos y a retrasos en el desarrollo.
Responder a cambios sobre seguir un plan: El equipo de desarrollo podría estar dispuesto a responder a los cambios en los requisitos o en las condiciones, en lugar de seguir estrictamente un plan. Esto podría dificultar la planificación y la gestión del proyecto.

Ingeniería de software centrada en las personas sobre herramientas y procesos: El equipo de desarrollo podría centrarse en las personas que desarrollan el software, en lugar de las herramientas y los procesos utilizados. Esto podría conducir a un desarrollo menos eficiente o a la entrega de un producto de menor calidad.

Para evitar estos conflictos, el equipo de desarrollo de OptiEstudio debe tener una comprensión clara de los valores del Manifiesto Ágil y estar dispuesto a comprometerse con ellos. El equipo también debe establecer procesos y prácticas que permitan equilibrar los valores del Manifiesto con las necesidades del proyecto.

Posibles soluciones para los conflictos potenciales mencionados anteriormente:

Individuos e interacciones sobre procesos y herramientas: El equipo de desarrollo puede crear procesos y herramientas que apoyen la colaboración y la comunicación. Por ejemplo, el equipo puede utilizar un sistema de gestión de proyectos para rastrear el progreso del desarrollo y comunicarse con el cliente.
Software funcionando sobre documentación exhaustiva: El equipo de desarrollo puede crear documentación funcional que describa cómo funciona el software. Esta documentación puede ser desarrollada por el equipo de desarrollo o por el cliente.

Colaboración con el cliente sobre negociación contractual: El equipo de desarrollo puede trabajar con el cliente para desarrollar un contrato que sea flexible y que permita los cambios necesarios.
Responder a cambios sobre seguir un plan: El equipo de desarrollo puede establecer un plan que sea flexible y que permita los cambios necesarios. El equipo también puede desarrollar procesos para identificar y gestionar los cambios.
Ingeniería de software centrada en las personas sobre herramientas y procesos: El equipo de desarrollo puede centrarse en las personas que desarrollan el software, pero también puede utilizar herramientas y procesos que les ayuden a ser más eficientes y efectivos. Por ejemplo, el equipo puede utilizar herramientas de automatización para realizar tareas repetitivas.

El equipo de desarrollo de OptiEstudio debe considerar cuidadosamente estas soluciones y elegir las que sean más adecuadas para el proyecto.

De las metodologías ágiles (Scrum, XP, Kanban, Design sprint, etc.), selecciona dos de ellas e identifica al menos tres de los principios de agilidad de software presentes en dichas metodologías.
Explique también de qué manera pueden integrarse esos principios de agilidad en su proyecto de
equipo, suponiendo que sigue una metodología ágil

Metodologías ágiles seleccionadas: Scrum y Kanban

Principios de agilidad presentes en las metodologías seleccionadas:

Individuos e interacciones sobre procesos y herramientas: Ambas metodologías ponen un énfasis especial en las interacciones entre los miembros del equipo y los usuarios. En Scrum, esto se logra a través de las reuniones diarias del equipo, las reuniones de revisión del sprint y las reuniones de retrospectiva del sprint. En Kanban, esto se logra a través de la visualización del trabajo y la colaboración en tiempo real.

Software funcionando sobre documentación exhaustiva: Ambas metodologías se centran en la entrega de software funcional de forma regular. En Scrum, esto se logra a través de la entrega de incrementos de trabajo al final de cada sprint. En Kanban, esto se logra a través de la entrega de trabajo de forma continua.

Responder a cambios sobre seguir un plan: Ambas metodologías permiten cambios en los requisitos o en las condiciones del proyecto. En Scrum, esto se logra a través de las reuniones diarias del equipo, donde el equipo puede identificar y abordar los cambios necesarios. En Kanban, esto se logra a través de la visualización del trabajo y la colaboración en tiempo real.

Integración de los principios de agilidad en el proyecto de equipo:

El equipo de desarrollo puede integrar los principios de agilidad en su proyecto de equipo de varias maneras. A continuación, se presentan algunas sugerencias:

Facilitar la comunicación y la colaboración: El equipo puede organizar reuniones regulares para discutir el progreso del proyecto y obtener comentarios de los usuarios. También puede utilizar herramientas de colaboración para facilitar la comunicación entre los miembros del equipo.

Priorizar la entrega de software funcional: El equipo puede dividir el proyecto en incrementos pequeños y entregarlos de forma regular. También puede utilizar técnicas de desarrollo incremental para facilitar la entrega de software funcional.

Ser receptivo a los cambios: El equipo debe estar preparado para responder a los cambios en los requisitos o en las condiciones del proyecto. También puede establecer procesos para identificar y gestionar los cambios.

En el caso específico del proyecto OptiEstudio, los principios de agilidad pueden integrarse de la siguiente manera:

El equipo puede organizar reuniones diarias del equipo para discutir el progreso del desarrollo y obtener comentarios de los usuarios. Estas reuniones pueden ayudar a garantizar que el equipo esté trabajando en las tareas más importantes y que el software satisfaga las necesidades de los usuarios.

El equipo puede utilizar herramientas de colaboración, como Slack o Jira, para facilitar la comunicación entre los miembros del equipo. Estas herramientas pueden ayudar a garantizar que el equipo esté trabajando de manera coordinada y que todos estén al tanto de los últimos desarrollos.

El equipo puede dividir el proyecto en incrementos pequeños, como sprints de dos semanas. Esto ayudará a garantizar que el equipo esté entregando software funcional de forma regular y que los usuarios puedan proporcionar comentarios a medida que el proyecto avanza.

El equipo puede utilizar técnicas de desarrollo incremental, como la programación en pares, para facilitar la entrega de software funcional. Estas técnicas pueden ayudar a garantizar que el software sea de alta calidad y que se cumplan los requisitos de los usuarios.

El equipo debe estar preparado para responder a los cambios en los requisitos o en las condiciones del proyecto. Esto puede hacerse mediante la creación de un proceso para identificar y gestionar los cambios.

Bibliografía:

Instituto Nacional de Estándares y Tecnología de los Estados Unidos (NIST). Framework for Improving Critical Infrastructure Cybersecurity. NIST Special Publication 800-53 Revision 5, 2022.

National Institute of Standards and Technology (NIST). Security Controls Catalog for Federal Information Systems and Organizations. NIST Special Publication 800-53 Revision 5, 2022.

OWASP Foundation. The OWASP Top 10 Project. 2022.

Schwaber, K., & Sutherland, J. (2022). The Scrum Guide. Scrum Alliance.

Anderson, D. J. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Addison-Wesley.

Agile Alliance. (2001). Manifesto for Agile Software Development.

Sutherland, J. (2014). Scrum: The Art of Doing Twice the Work in Half the Time. Crown Business.

Beck, K. (2000). Extreme Programming Explained: Embrace Change. Addison-Wesley.

Cockburn, A. (2001). Agile Software Development: The Cooperative Game. Addison-Wesley.