Metodologías ágiles - Ejercicio 13

a) Utiliza los 12 principios XP para organizar el cursado y aprobación de este año.

Qué:

- Aplica a cada principio a las distintas etapas del proceso de cursado.

Cómo:

- Para cada principio indique 3 ejemplos de aplicación que creas conveniente para tu situación con estudiante
- Agrega un ejemplo más que pueda servir para estudiantes en general

b) Utiliza un ML (modelo de lenguaje) para realizar una tabla de relación de principios

a)

1. Sit together

Qué: Trabajar en equipos colocados físicamente juntos para facilitar la comunicación y la colaboración.

Cómo:

- Proyectos en grupo: Organizar sesiones de estudio en las que los compañeros se sienten juntos para discutir y resolver problemas.
- Trabajo colaborativo: Participar en actividades grupales en el aula donde los estudiantes puedan interactuar y compartir ideas fácilmente.
- Sesiones de tutoría: Programar reuniones regulares con compañeros de clase para revisar y discutir el material de estudio.
- Ejemplo general: Establecer espacios de trabajo compartidos en la biblioteca o en áreas de estudio para fomentar la colaboración entre estudiantes.

2. Whole Team

Qué: Involucrar a todo el equipo, incluidos los estudiantes, profesores y personal de apoyo, en todas las etapas del proceso de cursado.

Cómo:

- Colaboración interdisciplinaria: Trabajar en proyectos interdisciplinarios que involucren a estudiantes de diferentes áreas de estudio.
- Feedback: Solicitar y considerar la retroalimentación de profesores y compañeros en el proceso de aprendizaje.
- Participación activa: Fomentar la participación activa de todos los miembros del equipo en discusiones y actividades académicas.
- Ejemplo general: Implementar programas de tutoría entre estudiantes de diferentes niveles para fomentar la colaboración y el aprendizaje mutuo.

3. Informative Workspace

Qué: Crear un entorno de trabajo que proporcione información clara y útil sobre el progreso y el estado del proyecto.

Cómo:

- Tableros de tareas: Utilizar tableros kanban o software de gestión de proyectos para visualizar y rastrear el progreso de las tareas académicas.
- Calendario compartido: Mantener un calendario compartido con fechas de entrega y eventos importantes para mantener a todos los miembros del equipo informados.
- Documentación: Mantener registros detallados de reuniones, decisiones y avances del proyecto para facilitar la comunicación y la colaboración.
- Ejemplo general: Establecer un sistema de seguimiento de proyectos en línea donde los estudiantes puedan actualizar y compartir su progreso con el equipo.

4. Energized Work

Qué: Fomentar un ambiente de trabajo energizado y motivador que impulse el rendimiento y la creatividad.

Cómo:

- Variedad de actividades: Incorporar una variedad de actividades y métodos de enseñanza para mantener el interés y la motivación de los estudiantes.
- Reconocimiento: Reconocer y celebrar los logros y avances del equipo para mantener altos niveles de motivación y compromiso.
- Descansos programados: Programar descansos regulares durante las sesiones de estudio para mantener la energía y la concentración.
- Ejemplo general: Organizar eventos sociales y actividades extracurriculares para promover el compañerismo y el bienestar emocional de los estudiantes.

5. Pair Programming

Qué: Trabajar en parejas para mejorar la calidad del código y fomentar el intercambio de conocimientos.

Cómo:

- Estudio en parejas: Realizar sesiones de estudio en parejas para discutir y resolver problemas juntos.
- Revisiones de código: Realizar revisiones de código en parejas para identificar errores y mejorar la calidad del trabajo.
- Proyectos en equipo: Trabajar en proyectos grupales asignando parejas de programación para colaborar en la codificación y el diseño.
- Ejemplo general: Implementar programas de tutoría entre estudiantes de diferentes niveles para fomentar el trabajo en equipo y el intercambio de conocimientos.

6. Stories

Qué: Utilizar historias de usuario para describir los requisitos desde la perspectiva del usuario y facilitar la planificación y ejecución del proyecto.

Cómo:

- Casos de uso: Desarrollar historias de usuario que describan las funcionalidades y requisitos del proyecto desde la perspectiva del estudiante.
- Priorización: Priorizar las historias de usuario según su importancia y valor para el proceso de aprendizaje.
- Iteraciones: Desarrollar y completar historias de usuario en iteraciones cortas y frecuentes para obtener retroalimentación temprana.
- Ejemplo general: Utilizar historias de usuario para describir las funcionalidades y requisitos de una plataforma en línea de aprendizaje para estudiantes de diversas disciplinas.

7. Weekly Cycle

Qué: Realizar entregas frecuentes y regulares de trabajo completado para obtener retroalimentación rápida y adaptarse a los cambios.
Cómo:

- Entregas semanales: Establecer plazos semanales para la entrega de trabajos, tareas o proyectos para mantener un flujo constante de progreso.
- Revisión de avances: Revisar y discutir el progreso del proyecto con el equipo de estudio al final de cada semana para identificar áreas de mejora.
- Ajustes continuos: Adaptar y ajustar el plan de estudios y las estrategias de aprendizaje según la retroalimentación recibida durante las revisiones semanales.
- Ejemplo general: Establecer plazos semanales para la entrega de informes de avance en proyectos de investigación en el ámbito académico.

8. Quarterly Cycle

Qué: Planificar y revisar los objetivos y prioridades a largo plazo en ciclos trimestrales para mantener el enfoque y la dirección del equipo.
Cómo:

- Planificación trimestral: Establecer metas y objetivos específicos para el trimestre en términos de rendimiento académico y desarrollo personal.
- Revisión trimestral: Evaluar el progreso hacia los objetivos establecidos y ajustar las estrategias y planes según sea necesario.
- Celebración de logros: Reconocer, celebrar los logros e hitos alcanzados al final de cada trimestre para mantener la motivación y el compromiso del equipo.
- Ejemplo general: Establecer objetivos trimestrales de aprendizaje y desarrollo para mejorar las habilidades y competencias de los estudiantes en áreas específicas.

9. Slack

Qué: Mantener un margen de tiempo y recursos adicionales para hacer frente a imprevistos y cambios inesperados.
Cómo:

- Tiempo de reserva: Asignar tiempo adicional en el cronograma de estudios para hacer frente a posibles retrasos o contratiempos.
- Recursos extra: Disponer de recursos adicionales, como materiales de referencia o apoyo técnico, para abordar problemas inesperados durante el proceso de aprendizaje.
- Flexibilidad: Ser flexible en la planificación y el enfoque del estudio para adaptarse a cambios repentinos en las circunstancias.
- Ejemplo general: Reservar tiempo adicional en el cronograma de un proyecto de investigación para abordar posibles problemas técnicos o experimentales.

10. Ten-Minute Build

Qué: Automatizar el proceso de construcción y despliegue para minimizar el tiempo dedicado a estas tareas.

Cómo:

- Automatización: Utilizar herramientas de automatización de construcción y despliegue para agilizar el proceso de desarrollo de software.
- Pruebas continuas: Integrar pruebas automáticas en el proceso de construcción para identificar errores de manera temprana.
- Despliegue continuo: Implementar un flujo de trabajo de despliegue continuo para entregar rápidamente nuevas funcionalidades a los usuarios finales.
- Ejemplo general: Utilizar herramientas de integración continua y entrega continua (CI/CD) en el desarrollo de aplicaciones web para garantizar la calidad y la rapidez en las entregas de software.

11. Continuous Integration

Qué: Integrar cambios de código frecuentemente en un repositorio compartido para detectar y corregir errores de manera rápida y eficiente.

Cómo:

- Integración regular: Realizar integraciones de código en un repositorio compartido varias veces al día para mantener actualizado el código base.
- Pruebas automatizadas: Ejecutar pruebas automáticas después de cada integración para identificar y solucionar errores de manera inmediata.
- Retroalimentación rápida: Proporcionar retroalimentación inmediata a los desarrolladores sobre la integridad del código y la estabilidad del sistema.
- Ejemplo general: Implementar un flujo de trabajo de integración continua en un proyecto de desarrollo de software para garantizar la estabilidad y la calidad del código.

12. Test-first programming

Qué: Escribir pruebas automáticas antes de escribir el código de producción para garantizar su calidad y funcionalidad.

Cómo:

- Desarrollo dirigido por pruebas (TDD): Escribir pruebas automatizadas para los requisitos de una funcionalidad antes de implementarla.
- Cobertura de pruebas: Asegurarse de que todas las funcionalidades estén respaldadas por pruebas automatizadas para garantizar su correcto funcionamiento.
- Refactorización segura: Realizar cambios en el código con confianza, sabiendo que las pruebas automatizadas garantizan la integridad y la funcionalidad del sistema.
- Ejemplo general: Aplicar el desarrollo dirigido por pruebas en el desarrollo de aplicaciones móviles para garantizar su fiabilidad y estabilidad.

b)

Principio XP	**Relación con la materia de metodologías ágiles**
Sit together	La materia fomenta el trabajo en equipo y la colaboración estrecha entre los miembros del equipo. Se enfatiza la importancia de la comunicación y la interacción frecuente entre los integrantes del equipo de desarrollo, así como la eliminación de barreras que puedan dificultar la comunicación efectiva.
Whole team	La materia promueve la participación activa de todos los miembros del equipo en todas las etapas del proceso de desarrollo de software. Se busca la integración y colaboración de diferentes roles y habilidades para lograr un resultado óptimo. Se evita la segmentación excesiva del trabajo y se fomenta la responsabilidad compartida y la toma de decisiones conjunta.
Informative workspace	La materia destaca la importancia de contar con un entorno de trabajo adecuado y que proporcione la información necesaria. Se promueve la visualización clara del progreso del proyecto, el estado de las tareas y la comunicación efectiva de la información relevante para todos los miembros del equipo.
Energized work	La materia busca fomentar un ambiente de trabajo motivador y enérgico. Se promueve la satisfacción y el compromiso de los miembros del equipo a través de la asignación de tareas interesantes y desafiantes, así como la creación de un entorno de trabajo estimulante y de apoyo mutuo.
Pair programming	La materia enseña y promueve la práctica de la programación en pareja. Se enfatiza la importancia de la revisión continua del código, el aprendizaje mutuo, la detección temprana de errores y la mejora de la calidad del software a través de la colaboración entre los miembros del equipo.

Stories	La materia se enfoca en la creación y gestión de historias de usuario como unidad básica de trabajo. Se enseña a descomponer los requisitos en historias pequeñas y manejables, priorizarlas y planificar su implementación en ciclos de trabajo. Se busca la entrega de valor incremental y la adaptabilidad a medida que se obtiene retroalimentación del cliente.
Weekly cycle	La materia promueve la adopción de ciclos de trabajo semanales, donde se planifican, implementan y entregan incrementos de software funcionales en cortos periodos de tiempo. Se busca obtener resultados tangibles de forma regular y obtener retroalimentación temprana para ajustar el enfoque y las prioridades.
Quarterly cycle	La materia también considera ciclos de trabajo más largos, como los trimestrales, donde se pueden planificar y ejecutar actividades de mayor envergadura, como la integración de componentes o la mejora de la infraestructura. Se busca un equilibrio entre la entrega rápida y la planificación a largo plazo.
Slack	La materia reconoce la importancia de contar con tiempo libre o margen de maniobra para hacer frente a imprevistos, mejorar la calidad del software y permitir la experimentación y la innovación. Se busca evitar la sobrecarga de trabajo constante y permitir un equilibrio entre la productividad y la salud del equipo.
Ten-minute build	La materia promueve la automatización de la construcción del software y la reducción del tiempo necesario para compilar, integrar y probar el código. Se busca minimizar los tiempos de espera y maximizar la eficiencia del proceso de desarrollo.
Continuous integration	La materia destaca la importancia de la integración continua, donde se fusionan y verifican los cambios de código realizados por los miembros del equipo de forma regular y automatizada. Se busca detectar y solucionar problemas de integración rápidamente, así como mantener una base de código estable y confiable.
Test-first programming	La materia enfatiza la práctica de escribir pruebas automatizadas antes de desarrollar el código. Se promueve la implementación de pruebas unitarias y de integración desde el inicio del desarrollo, lo que permite detectar errores tempranamente, mejorar la calidad del software y facilitar la refactorización del código.