Ingeniería de Software

-

Dr. Christian Quesada López

Bach. Erik Kuhlmann Salazar

-

Evaluación técnica 1

-

Fernando Arce Castillo

Archibald Emmanuel Carrion Claeys

Fundamentos de la ingeniería de software, análisis y discusión (9%). Una página para la respuesta de cada pregunta.

1. Como ingeniero/ingeniera de software. El equipo de desarrollo al que usted pertenece en su organización utiliza la metodología Scrum como la metodología para el desarrollo de software. Actualmente la gerencia técnica está planeando cambiar la metodología a XP. Describa en detalle un caso a favor de continuar con la metodología Scrum, y un caso a favor de cambiar a la metodología XP. Para su respuesta considere los fundamentos de cada metodología, las similitudes y las diferencias, los roles, sesiones y artefactos, y las reglas del proceso de cada uno de ellos. Finalmente, de una recomendación justificada técnicamente sobre la selección de la metodología a la gerencia técnica de su organización (4.5%).

Nos parece importante empezar diciendo que ambas metodologías son muy buenas opciones que permiten llegar a buenos rendimientos y que ambas ya fueron probadas muchas veces. Tienen ventajas, y desventajas, por lo que a continuación hablaremos de las ventajas de scrum, y después de XP.
Empezamos dando nuestras razones de porque scrum es una muy buena opción de desarrollo. Scrum presenta muchas ventajas, de las cuales la principal y la más conocida, es una metodología muy flexible y que permite adaptarse fácilmente a las situaciones que aparecen a lo largo del desarrollo. Esas situaciones pueden ser de igual manera requisitos del cliente como nuevas tecnologías que aparecen en el mercado.
Scrum se basa mucho en reuniones, que sea reuniones dailies o reuniones entre embajadores, es algo muy importante dentro de las reglas de scrum, por lo que dependiendo de la organización de los equipos puede llegar a ser muy valioso para todos los teams de desarrollo. Aunque las reuniones pueden llegar a ser un arma de doble filo, como lo mencionamos cuando mencionamos las ventajas de la metodología XP.
Después investigar en clase scrum y aplicarlo a lo largo de la clase de PI parecería que scrum tiende a ser un sistema más robusto y más escalable, donde XP parecería ser una metodología óptima para un team concentrado únicamente en desarrollo y más pequeña.

Por otro lado, la metodología XP presenta muchas ventajas sobre scrum. Scrum se basa muchísimo en reuniones, las cuales si no son bien manejadas pueden llegar a hacer que los desarrolladores pierden tiempo muy valioso. Como mencionado en el articulo Daily Stand Up Meeting de extremeprogramming.org: “A large amount of developer time is wasted to gain a trivial amount of communication. Having many people attend every meeting drains resources from the project and also creates a scheduling nightmare.”.
Como ventaja adicional, XP tiene como pilar principal producir código de muy alta calidad, como lo indica el nombre (extreme programming). Por esa misma razón, XP presenta varias metodologías de trabajo que permite producir código de calidad, tal como pair-programming o mob-programming. Según las reglas de XP, programmer en pareja permite crear tanto código que si los dos programadores trabajan de manera independiente pero con un resultado de mayor calidad: “All code to be sent into production is created by two people working together at a single computer. Pair programming increases software quality without impacting time to deliver. It is counter intuitive, but 2 people working at a single computer will add as much functionality as two working separately except that it will be much higher in

quality”. Por lo que dependiendo de la cultura de la empresa y de la productividad real que sale del trabajo en pair programming, puede llegar a no ser tan económicamente factible.

2. Como ingeniero/ingeniera de software. El equipo de desarrollo al que usted pertenece en su organización no ha utilizado hasta el momento arquitecturas limpias, código limpio y principios SOLID en el desarrollo de las aplicaciones. Explique los principales beneficios (con ejemplos) de cada una de estas prácticas e indique cuál es su recomendación para implementar la utilización de estas como parte de su proceso de desarrollo. Sea puntual mencionando cada concepto, las ventajas que ofrece y porqué (con ejemplos) (4.5%).

Para muchos programadores, usar arquitecturas limpias, escribir código limpio y seguir los principios SOLID puede parecer abrumador inicialmente. Sin embargo, aunque los que no están acostumbrados a trabajar con estas tecnologías pueden llegar a durar más tiempo produciendo código al principio, pero rápidamente se darán cuenta de que están creando código de mayor calidad. Este código resultará ser más escalable, mantenible y, en general, más fácil de modificar en el futuro. A largo plazo, aplicar estas prácticas de desarrollo hará que el team de desarrollo tenga mejor organización y en general hará mejor código.

La arquitectura limpia se basa en una serie de conceptos que permiten llegar a tener una estructura general del código lógica en vez de ser un aglomerado de código arbitrario. Una de esas técnicas consiste en separar las tareas necesarias para la realización de un proyecto en diferentes posibles “capas” que pueden comunicarse entre ellas o no dependiendo de la arquitectura. Esas capas permiten llegar a un mayor nivel de abstracción que permite a grupos grandes de trabajo no preocuparse por cómo funciona el código dentro de las demás capas. Para ejemplificar las ventajas de una arquitectura limpia, al trabajar con un team con desarrollador frontend y backend pueden gracias a la nueva arquitectura trabajar de manera más eficiente sin tener que pasar tiempo entendiendo códigos de otras capas. Nada más usando el API desarrollado por parte del team frontend puede conectarse con backend.

El código limpio es un término muy grande que engloba una multitud de conceptos, que consiste en escribir código claro, bien comentado, fácil de entender y que tiende a ser el más simple posible separando cada clase en archivos diferentes y así. El código limpio se opone a lo llamado “code smells”, término que usamos para referirnos a un código “malo” que tal vez es funcional, pero poco mantenible, mal escrito, no óptimo, o que no cumple con alguna de las convenciones de clean code establecidas por el team. Para ejemplificar las ventajas de usar clean code, si mañana la empresa decide agregar nuevos miembros al team de desarrollo o agregar otro team por completo. Si ese nuevo team de una manera o otra tiene que usar el código anteriormente hecho, y no es código limpio entonces los nuevos desarrolladores van a durar más tiempo entendiendo el código, ya que no presenta una buena consistencia ni buenas prácticas. En el caso contrario, si el código está bien hecho y sigue los reglamentos de código limpio entonces los nuevos desarrolladores podrán más fácilmente usarlo como base para seguir trabajando.

Para terminar, los principios SOLID son un grupo de 5 principios vitales en programación que permiten llegar a una mayor calidad de código, estos principios son:

- responsabilidad unitaria (cada clase tiene que funcionar por sí mismo realizando una tarea muy específica).

- Abierto/cerrado consiste en decir que una clase tiene que ser abierta para ser usada por otras clases pero cerrada en ser modificada por otras clases.
- substitución de Liskov, consiste en decir que si existe una clase que tiene clases derivadas basada en uno mismo entonces tiene que poder intercambiar de rol con la clase derivada en cualquier momento.
- El principio de segregación de interfaz, consiste en plantear una solución que propone una interfaz especial para el cliente dado, una interfaz mínima que no obliga el usuario manejar interfaz que no quiere o ocupa.
- Principio de inversión de dependencias, los módulos de alto nivel no deben depender de los módulos de bajo nivel.

Para ejemplificar la importancia de apoyarse en los principios SOLID, si el código no responde al primer principio de responsabilidad única, el código se vuelve pesado y complicado de modificar. Por ejemplo, si tenemos una clase que permite agregar una nueva entidad en la base de datos, pero la misma clase también lo puede modificar, y de paso crear otra instancia diferente, nos damos cuenta que no es un código simple en entender, tenemos una mezcla de responsabilidad. Al no respetar ese principio podemos llegar a tener un “código espagueti” donde las responsabilidades no están bien definidas.

Requerimientos (15%)

1. Especifique al menos un tema, un epic y tres historias de usuario debidamente documentadas sobre cualquiera de las funcionalidades del sistema (seleccionadas por prioridad según su criterio profesional). Describa al menos dos criterios de aceptación por historia de usuario (uno funcional y uno no funcional). Justifique el cumplimiento de los criterios INVEST para cada una de las historias (8%).

***Tema**:* Mejorar la visibilidad de las carreras y sus componentes dentro de la UCR.

***Epic**:* Como usuario administrador de la UCR quiero poder manejar las carreras y sus componentes mediante una página web.

***Historia de usuario 1*:** Como administrador de la página quiero poder agregar una nueva carrera para apegarse a la oferta académica de la Universidad.

- El administrador debe poder acceder a la función de agregar una nueva carrera desde la página web.
- Agregar la carrera debe ser un proceso fácil de realizar para alguien que no conoce de tecnología.

Criterios INVEST

- Esta historia de usuario es independiente de las demás tareas porque representa un conjunto de funcionalidades claras y delimitadas: crear una nueva carrera, no depende ni de poder agregar componentes a una carrera ni tampoco de poder listar o buscar carreras y componentes.
- Se puede negociar con cuáles aspectos se puede crear una carrera, puede fácilmente modificarse más tarde.

- Es muy valioso para la empresa y el sistema poder agregar una carrera ya que permite a la universidad mantener al día sus ofertas académicas.
- Es muy estimable ya que tiene criterios de aceptación establecidos y tareas claras.
- Es una granularidad pequeña ya que crear una carrera es una tarea muy definida que solo tiene como objetivo agregar al sistema una nueva carrera sin componentes.
- Es testeable ya que sabemos que tiene que realizar la historia de usuario y mediante pruebas podemos verificar que la creación de carrera funciona correctamente.

Historia de usuario 2: Como administrador de la página quiero poder agregar contenido a una carrera ya existe para representar todos los componentes de una carrera.

- El administrador debe poder acceder a la función de agregar contenido de una carrera ya existente desde la página web.
- Agregar un nuevo componente a una carrera debe ser intuitivo y fácil de usar para cualquier usuario administrador que no sabe de tecnología.

Criterios INVEST

- Esta historia de usuario es independiente de las demás tareas porque representa un conjunto de funcionalidades claras y delimitadas: crear un nuevo componente para una carrera ya existente, no depende ni de crear carrera ni tampoco listar/buscar carreras.
- Se puede negociar con cuáles características se puede crear un nuevo componente, puede fácilmente modificarse más tarde.
- Es muy valioso para la empresa y el sistema poder agregar un componente a una carrera ya que permite a la universidad mantener al día sus ofertas académicas.
- Es muy estimable ya que tiene criterios de aceptación establecidos y tareas claras.
- Es una granularidad pequeña ya que crear un componente de una carrera es una tarea muy definida que solo tiene como objetivo agregar al sistema un nuevo componente para una carrera ya existente.
- Es testeable ya que sabemos que tiene que realizar la historia de usuario y mediante pruebas podemos verificar que funciona agregar un nuevo componente.

Historia de usuario 2*:* Como administrador de la página quiero poder listar las carreras y sus contenidos para tener una mejor visión global de la universidad y sus componentes.

- El administrador debe poder acceder a la lista de carreras y sus contenidos desde la página web.
- La página que muestra la lista de carreras y sus contenidos debe mostrar todas las informaciones necesarias sin que la interfaz se vea demasiado cargada.

Criterios INVEST

- Esta historia de usuario es independiente de las demás tareas porque representa un conjunto de funcionalidades claras y delimitadas: listar las carreras y sus informaciones, no ocupa ni poder crear carreras o componentes para que esta historia funcione.
- Se puede negociar cuáles datos mostrar en pantalla, se puede fácilmente modificarse más tarde según preferencias del PO y clientes.
- Es muy valioso para la empresa y el sistema poder facilmente ver cuales son las carreras de la universidad y las informaciones de cada una.
- Es muy estimable ya que tiene criterios de aceptación establecidos y tareas claras.

- Es una granularidad pequeña ya que listar las carreras es una tarea muy definida que solo tiene como objetivo imprimir en una tabla las carreras.
- Es testeable ya que sabemos que tiene que realizar la historia de usuario y mediante pruebas podemos verificar que el listado funcione correctamente.

2. Especifique tres requerimientos no funcionales prioritarios para el sistema, clasifíquelos y cuantifíquelos. Justifique la selección (7%).

Prioridad 1 - Mantenimiento del sistema
Nuestra aplicación tiene que presentar códigos y arquitectura de alta calidad para asegurar facilidad de mantenimiento y mejoramiento en las próximas implementaciones.
Por lo que usaremos conventions de clean-code y arquitectura limpia como lo visto en clases.

Prioridad 2 - Usabilidad de la plataforma
La plataforma debe ser muy fácil de usar para cualquier usuario administrator.
El usuario final no tiene que conocer mucho de tecnología para lograr usar la aplicación.
La página es orientada hacia una población usuaria de sistema administrativos de la universidad de Costa Rica, no una población tecnologica.

Prioridad 3 - Escabilidad del sistema
Nuestra aplicación debe ser escalable y poder manejar muchos datos que sean grandes cantidades de carreras y de componentes, como grandes cantidades de usuarios que gusten ver listado de carreras y componentes.