Universidad ORT Uruguay

Facultad de Ingeniería

Informe Final Académico

Ingeniería de software ágil 2

Nahuel Larenas – 260079

Sofía Piñeyro – 243498

Florencia Varela – 254005

Profesores: Alvaro Ortas - Carina Fontan

Grupo: M7B

Junio 2023

Tabla de contenidos

Reflexiones sobre lo aprendido

Esta sección presenta un breve resumen de los objetivos planteados en la rúbrica y una reflexión sobre lo aprendido al cumplir con estos.

Definición de marco general de KANBAN

Durante la definición del marco general de Kanban, aprendimos la importancia de dividir las actividades de manera clara para asegurar una ejecución efectiva y lograr los resultados deseados. Además, destacamos la asignación clara de responsabilidades, el uso de herramientas como Github y las prácticas de BDD, que nos ayudaron a mantener un seguimiento adecuado de los requisitos y garantizar la calidad del código.

Sin embargo, también identificamos áreas de mejora. En algunas etapas, encontramos desafíos en cuanto a la estimación de esfuerzo requerido para ciertas actividades. En ocasiones, agrupamos tareas que pensamos que eran rápidas o simples con otras actividades, en lugar de considerarlas como actividades de Kanban independientes. Al evaluar el esfuerzo requerido para cada actividad y considerar la posibilidad de desglosar tareas que podrían funcionar mejor como actividades individuales, tendríamos una planificación más precisa y una asignación de recursos más efectiva.

Definición del proceso de ingeniería

Durante la definición del proceso de ingeniería utilizando BDD y Kanban, aprendimos la importancia de definir criterios de aceptación antes de la implementación. Valoramos la implementación de BDD y la utilización de Kanban como prácticas positivas que debemos mantener. Sin embargo, reconocemos que debemos mejorar en la comunicación y colaboración durante la definición de los criterios. En general, estos aprendizajes nos permitirán abordar futuros proyectos de manera más efectiva y lograr resultados de alta calidad.

Definición de roles del equipo

Durante la definición de roles de equipo, reconocimos la importancia de tener un Product Owner, Scrum Master, desarrolladores y testers en nuestro equipo. Sin embargo, no utilizamos el rol de diseñador que se planteó en la primera entrega (ver definición de roles). Para mejorar, debemos implementar reuniones regulares para sincronizar esfuerzos y mejorar la comunicación dentro del equipo. Además, en algunas instancias y por razones de fuerza mayor nos vimos obligados a rotar los roles, esto generó algunas confusiones a lo largo del proyecto.

Análisis de deuda técnica

Durante el análisis de la deuda técnica, implementamos herramientas como StyleCop Analyzers y ESLint, así como pruebas unitarias y pruebas exploratorias, las cuales resultaron ser útiles en nuestro proyecto. A través de estas prácticas, logramos identificar y abordar problemas de calidad y mantenibilidad del código de manera efectiva, por lo que consideramos que se puede implementar un monitoreo continuo de la calidad del código.

Retrospectiva en método DAKI

La retrospectiva en el método DAKI es una buena manera de mejorar como equipo cuando todos los miembros están comprometidos en aprovechar al máximo la reunión. Permite una reflexión profunda y estructurada sobre las prácticas y procesos del equipo, fomentando la identificación de áreas de mejora y la implementación de acciones correctivas. Sin embargo, es importante tener en cuenta que existen otros formatos de retrospectiva igualmente efectivos y más dinámicos. Estos formatos pueden incluir actividades más interactivas, como juegos, ejercicios de improvisación o técnicas visuales, que ayudan a mantener el interés y la participación activa de todos los miembros. Al explorar diferentes enfoques, se abre la posibilidad de descubrir nuevas perspectivas y enriquecer el aprendizaje del equipo.

Creación del tablero

La estructura del tablero debe reflejar el flujo de trabajo, desde la captura de requisitos hasta la entrega del producto final. Además, observamos que es esencial limitar la cantidad de trabajo en progreso, lo que ayuda a evitar la sobrecarga y el bloqueo de tareas. Asimismo, se valoró la importancia de fomentar la colaboración y la transparencia, permitiendo que todo el equipo tenga visibilidad sobre el estado de las tareas y las dependencias entre ellas. Por último, se detectó la necesidad de realizar ajustes continuos en el tablero de Kanban en función de la retroalimentación y los cambios en el proyecto.

Configuración del pipeline

Establecer un pipeline eficiente y sólido es crucial para agilizar y optimizar el flujo de desarrollo de software. La configuración adecuada del pipeline permite una entrega continua y una integración sin problemas, lo que acelera la velocidad de entrega y reduce el riesgo de errores. Además, se identificó la importancia de contar con pruebas automatizadas y revisiones de código en etapas cruciales del pipeline para garantizar la calidad del software. La visibilidad y transparencia en el pipeline son fundamentales para el seguimiento del progreso y la identificación temprana de posibles cuellos de botella. También se apreció la necesidad de adaptar y ajustar el pipeline de acuerdo con las necesidades cambiantes del proyecto y el equipo.

Reparación de ambos bugs utilizando TDD

Adoptar TDD como enfoque para corregir los bugs no solo ayuda a solucionar los problemas actuales, sino que también previene la aparición de futuros errores. Al seguir el ciclo de TDD, que implica escribir pruebas antes de implementar las correcciones, se obtiene una comprensión más clara de los requisitos y comportamiento esperado del sistema. Además, se encontró que el uso de pruebas automatizadas proporciona un medio confiable para verificar la funcionalidad corregida y mantener la integridad del código.

Realización de una revisión de los bugs con el PO.

Aprendimos que la revisión con el PO es crucial para garantizar la satisfacción del cliente y la alineación con los requisitos del producto. Al involucrar al PO en la revisión de los bugs

solucionados, se obtiene una visión clara de las expectativas del cliente y se asegura que los problemas reportados hayan sido abordados de manera efectiva.

Creación de escenarios de prueba en BDD para las nuevas funcionalidades

La creación de escenarios de prueba en BDD para nuevas funcionalidades es una estrategia efectiva para garantizar la calidad del software. Durante este proceso, se destacó la importancia de definir casos de prueba específicos que abarquen todos los aspectos relevantes de la nueva funcionalidad. Estos escenarios de prueba ayudan a identificar y validar de manera exhaustiva los diferentes comportamientos y resultados esperados de la funcionalidad implementada.

Desarrollo del front-end y back-end (en dicho orden) de las nuevas funcionalidades

Aprendimos que al seguir este orden de desarrollo, es muy importante la coordinación entre los desarrolladores, ya que de no contar con la misma información y llegar a acuerdos de por ejemplo *naming*, esto lleva a realizar un retrabajo en alguna de las dos partes adaptándose a la otra, principalmente si se pretende trabajar paralelamente. Además, al completar primero el frontend, ese user story/bug va a quedar bloqueado hasta que el backend esté pronto, ya que sin él no podrá entrar en etapa de testing.

Automatización del test exploratorio (funcional con Selenium) de ambos bugs y ambas nuevas funcionalidades.

Es una muy buena manera para garantizar el correcto funcionamiento del frontend, además, sirve como documentación de cómo realizar procesos esenciales del sistema de forma rápida y confiable. Es importante también realizar tests robustos los cuales no dependan de recursos ya creados, sino que sean independientes y de lo posible, no dejen en la base de datos recursos creados innecesariamente. Es decir, idealmente un test debería crear los recursos necesarios que desea testear, y una vez hechas las validaciones correspondientes, eliminar esos recursos.

Confección y análisis de métricas DevOps.

Vimos que es una buena manera de en números darse cuenta cuando hay potenciales aspectos a mejorar en el flujo de trabajo, ya que cuando tuvimos cuellos de botellas en el pipeline fue clara la diferencia, por ejemplo, en el flow efficiency. Sin embargo, es necesario que todos los miembros del equipo mantengan el tablero lo más fiable posible, estando las user stories y bugs en el estado que verdaderamente se encuentran, de lo contrario las métricas no van a dar valores que realmente aporten información valiosa para detectar posibles aspectos a mejorar en el flujo de trabajo del equipo.

Lecciones Aprendidas

Durante la realización del proyecto, identificamos valiosas lecciones aprendidas que pueden servir de guía para futuros proyectos similares. A continuación, se detallan algunas de estas lecciones, junto con consejos específicos sobre las metodologías y prácticas utilizadas.

Metodología KANBAN

- Si deben utilizar la metodología Kanban, entonces es crucial establecer y comunicar de manera efectiva las políticas de flujo de trabajo.

En ocasiones, nos resultó difícil seguir el flujo de trabajo establecido en KANBAN, lo que limitó nuestra capacidad de aprovechar plenamente la metodología. Por ejemplo, olvidamos actualizar las tareas en el tablero, lo que generó confusión y dificultades para dar seguimiento al progreso del proyecto. Aprendimos la importancia de la disciplina y la consistencia para mantener actualizado el tablero y aprovechar al máximo los beneficios de KANBAN.

Práctica de Desarrollo Dirigido por Pruebas (TDD)

- Si quieren escribir código de calidad fácilmente, entonces utilicen TDD.

Al usar TDD para desarrollar funcionalidades, nos dimos cuenta de que las pruebas nos obligaron a pensar detenidamente en los casos de usos y requisitos de las funcionalidades. Esto nos ayudó a comprender mejor el comportamiento que debía tener el código, asegurando así su calidad.

- Si deben usar TDD, entonces tengan en cuenta que se debe invertir un mayor esfuerzo inicial.

En relación a lo mencionado anteriormente, al aplicar TDD y tener que reflexionar detenidamente sobre los casos de uso y requisitos de cada funcionalidad antes de comenzar a desarrollar, nos enfrentamos a un esfuerzo adicional en las etapas iniciales del proceso.

Práctica de Desarrollo Dirigido por Comportamiento (BDD)

- Si quieren comprender detalladamente el comportamiento de una funcionalidad previo a desarrollarla, entonces deben usar BDD.

Durante el desarrollo de funcionalidades con BDD, observamos que al comenzar a escribir código teníamos un claro entendimiento de cómo debía comportarse cada funcionalidad. Esto nos facilitó el desarrollo del código y nos aseguró que la calidad del mismo era buena.

- Si quieren utilizar BDD, entonces debe haber una comunicación clara y colaboración entre el equipo y el PO.

Observamos que era necesario que la colaboración y la comunicación entre el equipo fuera buena para poder evitar malentendidos y aprovechar al máximo los beneficios de BDD.

- Si utilizan BDD, entonces tengan en cuenta que se debe invertir un mayor esfuerzo inicial.

Al igual que con TDD, utilizar BDD nos obliga a invertir un mayor esfuerzo al comienzo del proyecto.

Método DAKI (Definir, Aprender, Decidir, Hacer)

- Si quieren identificar áreas de mejora para poder tomar decisiones basadas en evidencia, entonces deben utilizar el método DAKi en las retrospectivas.

En las retrospectivas, observamos que al utilizar el método DAKI se nos facilitó la identificación de problemas y generación de soluciones concretas para su implementación.

Otros consejos

- Si tienen que enfrentarse a tecnologías nuevas, entonces hagan pair programming.

Al enfrentarnos a las nuevas tecnologías, descubrimos que nos fue muy útil hacer pair programming debido a que entendíamos más fácilmente el funcionamiento.

- Si realizan retrospectivas periódicas, entonces pueden reflexionar sobre el proceso y realizar mejoras

En las retrospectivas pudimos identificar acciones a mejorar y mantener, lo que nos permitió que el flujo de trabajo del equipo fuera mejor en cada entrega.

- Si mantienen una comunicación abierta y transparente, entonces el equipo trabaja mejor y se evitan malentendidos o tiempos muertos.

En las primeras entregas tuvimos algunas dificultades para mantener una comunicación continua, con lo que se produjeron algunos tiempos muertos. Luego de hablarlo en las retrospectivas correspondientes, nos dimos cuenta que al comunicarnos más frecuentemente, el trabajo fluía mejor y más eficientemente.

Gestión del proyecto

Actividades realizadas y productos entregados

El proyecto se dividió en cuatro etapas principales. En la primera etapa, que duró dos semanas, establecimos el marco general de KANBAN y creamos la primera versión del proceso de ingeniería. Realizamos análisis de deuda técnica según la guía en el anexo, definimos los roles del equipo y elaboramos guías sobre el uso del proceso y el repositorio. Además, registramos y clasificamos los issues en GitHub, presentamos un informe de avance y realizamos una retrospectiva con el Scrum Master. En esta, marcamos la importancia de la comunicación entre el equipo y nos comprometimos a mejorarla.

En la segunda etapa, de una semana de duración, nos enfocamos en la implementación práctica del proceso de ingeniería. Creamos el tablero en GitHub y configuramos el pipeline. Además, seleccionamos y reparamos dos bugs de alta severidad utilizando TDD, realizamos retrospectivas y revisiones con el Product Owner, y generamos evidencia de ejecución de casos de prueba. En la retrospectiva, valoramos la importancia de utilizar pair programming como herramienta al enfrentarnos a nuevas tecnologías.

En la tercera etapa, que abarcó tres semanas, actualizamos el proceso de ingeniería a una segunda versión basada en BDD y KANBAN. Creamos una nueva versión del tablero y configuramos el pipeline. Desarrollamos nuevas funcionalidades siguiendo un orden específico, realizamos la retrospectiva y actualizamos la documentación relacionada con el proceso, el tablero y el pipeline. En esta etapa, observamos que la eficiencia del equipo no fue muy elevada debido al cuello de botella generado al tener que terminar todo el frontend antes de comenzar el backend y destacamos la fluidez del trabajo en equipo así como la importancia de usar pair programming.

En la cuarta y última etapa, de una semana, automatizamos el test exploratorio funcional de los bugs reparados y las nuevas funcionalidades utilizando Selenium. Realizamos métricas DevOps, llevando a cabo una retrospectiva y documentamos los resultados. En esta, observamos que las métricas del equipo fueron buenas, y destacamos algunas acciones a tomar con el fin de mejorar la organización del trabajo en equipo.

Al final de cada etapa, presentamos informes de avance y realizamos videos de las retrospectivas y revisiones con el Scrum Master y el Product Owner, respectivamente.

Cantidad de issues reportados y cerrados

Según el reporte de issues, se reportaron un total de 43 bugs durante la primera entrega del proyecto. En la segunda entrega, se logró solucionar exitosamente dos de esos bugs, estos fueron seleccionados (cerrados) en base a su severidad y prioridad (ver criterio de selección).

Además de resolver los bugs, en la tercera entrega agregamos 4 mejoras, indicadas en la letra del proyecto. Estas mejoras incluyen la posibilidad de que el cliente realice la compra de snacks y el

administrador pueda hacer el mantenimiento de estos, en ambos casos se desarrolló tanto el frontend como el backend.

Hasta el momento reportamos 47 issues de los cuales 43 son issues (2 fueron solucionados), y 4 son mejoras que ya fueron entregadas (cerradas).

Métricas del proyecto

Se puede ver el informe de las métricas de cada entrega en los siguientes links: Entrega 2, Entrega 3 y Entrega 4.

Lead Time

El Lead Time es una métrica utilizada para evaluar la eficiencia y velocidad de entrega de un proceso o equipo de trabajo. Refiere al tiempo desde que se crea una tarea (se coloca en TO-DO) hasta que se finaliza la tarea (pasa a DONE). Se mide en alguna unidad de tiempo (horas, días, meses, etc).

Cálculo: Promedio de el siguiente cálculo aplicado a cada tarea:

Fecha de Entrega (pasa a DONE) - Fecha del Pedido (entra en TO DO)

	Lead time
Entrega 2	7 días
Entrega 3	24 días
Entrega 4	11 días

Cycle Time

El Cycle Time es una métrica utilizada para medir la duración promedio de una tarea o solicitud desde el momento en que se inicia hasta el momento en que se completa. Se enfoca en el tiempo real de trabajo activo en lugar del tiempo total transcurrido. Refiere al tiempo desde que se comienza una tarea (pasa a IN PROGRESS o primer paso del tablero luego de TODO) hasta que se finaliza la tarea (pasa a DONE). Se mide en alguna unidad de tiempo (horas, días, meses, etc).

Cálculo: Promedio de el siguiente cálculo aplicado a cada tarea:

Fecha de Entrega (pasa a DONE) – Fecha de Inicio (pasa a IN PROGRESS)

	Cycle Time
Entrega 2	2 días
Entrega 3	13 días

Entrega 4	2 días

Touch Time

El Touch Time es una métrica que se utiliza para medir el tiempo real de trabajo directo realizado en una tarea o solicitud. Se centra en el tiempo en el que se lleva a cabo el trabajo activo y directo, excluyendo cualquier tiempo de espera, demoras o interrupciones. Refiere al tiempo en el cual un ítem de trabajo fue realmente trabajado (o "tocado") por el equipo.

Cálculo: Promedio de el siguiente cálculo aplicado a cada tarea:

Cycle Time – Tiempo en espera por alguna razón

	Touch Time
Entrega 2	2 día
Entrega 3	1 días
Entrega 4	2 días

Esfuerzo total

El Esfuerzo Total es una métrica utilizada para medir la cantidad total de trabajo invertido en una tarea o proyecto. Esta métrica refleja la cantidad de tiempo, recursos y energía dedicados a la ejecución de una tarea o proyecto en su conjunto. Refiere a la cantidad de trabajo invertida por todas las personas que participan en una tarea, considerando solo el Touch Time.

Cálculo: suma del trabajo invertido en cada tarea (Touch Time)

	Esfuerzo total
Entrega 2	7:30 horas-persona
Entrega 3	23 horas-persona
Entrega 4	8:15 horas-persona

Flow Efficiency

El Flow Efficiency se refiere a la eficiencia con la que se está trabajando en el proyecto.

Cálculo: touch time/lead time

	Flow Efficiency
Entrega 2	28%

Entrega 3	7,7%
Entrega 4	18,18%

Throughput

El throughput se refiere a la cantidad de trabajo que se completa en un período de tiempo determinado.

Cálculo: Cantidad de trabajo/período de tiempo

	Throughput
Entrega 2	2 issues en 7 días
Entrega 3	2 issues en 24 días
Entrega 4	6 tareas en 11 días

Deployment Frequency

La frecuencia de implementación o "deployment frequency" es un indicador utilizado para medir con qué frecuencia se realiza la implementación o entrega de nuevas versiones de software a producción.

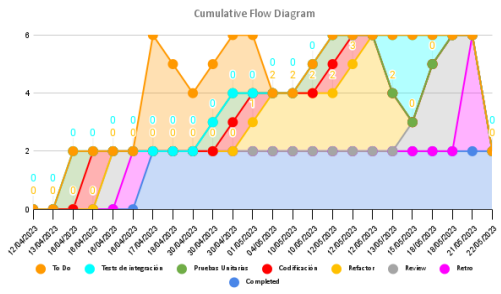
Cálculo: Número de implementaciones exitosas / Duración del período

	Deployment Frequency
Entrega 4	2 implementaciones exitosas/mes

Cumulative Flow Diagram

El diagrama de flujo acumulativo (Cumulative Flow Diagram, CFD) se utiliza para visualizar y analizar el flujo de trabajo de un proyecto a lo largo del tiempo. Proporciona una representación gráfica de cómo las tareas o elementos se mueven a través de diferentes etapas del proceso.

En la entrega 4, realizamos un diagrama que representa una estimación de la distribución de tareas a lo largo del tiempo.



Conclusiones sobre el proyecto

En conclusión, a lo largo de este proyecto hemos aprendido valiosas lecciones sobre la importancia de utilizar metodologías como Behavior Driven Development (BDD), Test Driven Development (TDD), DAKI y Kanban, así como definir claramente el proceso de ingeniería.

La implementación de BDD nos permitió definir criterios de aceptación desde el principio, lo cual aseguró la calidad del producto final y nos brindó una comprensión más profunda de los requisitos de cada funcionalidad. Por otro lado, la adopción de TDD nos impulsó a escribir código de calidad desde el principio, lo cual facilitó su mantenimiento y redujo la aparición de errores.

Además, la automatización de pruebas nos permitió agilizar y asegurar la calidad de nuestro proceso de desarrollo. Al implementar pruebas automatizadas, pudimos detectar rápidamente errores y problemas, lo que nos brindó una mayor confianza en la estabilidad y funcionalidad de nuestro software.

Asimismo, la metodología Kanban nos brindó una visualización clara del flujo de trabajo, lo que permitió un seguimiento transparente del progreso del proyecto y una adaptación continua a medida que surgían nuevas necesidades. Además, definir claramente el proceso de ingeniería nos brindó una estructura sólida y aseguró una asignación adecuada de responsabilidades en cada etapa.

Además, al medir métricas de DevOps, pudimos obtener información valiosa sobre la eficiencia y el rendimiento de nuestro equipo. Esto nos permitió identificar oportunidades de mejora y tomar acciones concretas para optimizar nuestra eficiencia y mejorar nuestro trabajo en equipo. Seguiremos utilizando estas métricas en futuros proyectos para asegurar un alto nivel de desempeño y colaboración en nuestro equipo.

En general, estas prácticas nos proporcionaron una mayor eficiencia en el desarrollo de software y mejoraron la calidad de nuestros productos. A través de este proyecto, hemos fortalecido nuestro entendimiento de estas metodologías y reafirmado su valor en la gestión de proyectos de ingeniería.