Universidad ORT

Ingeniería de Software Ágil 2

Entrega Final
Informe académico

Equipo 3

Federico Alonso
182999

Denise Souberville
223427

Horacio Ábalos
196991

02 de junio de 2023

Link al Repositorio

Tabla de contenidos

Introducción

El objetivo del proyecto obligatorio es aplicar los conceptos de DevOps en un proyecto existente mediante la implementación de prácticas de CI/CD, QA y testing automático. El proyecto se desarrollará en 5 entregas con duración variable y se utilizará el marco de gestión Kanban.

Desarrollo

Resumen de gestión del proyecto

En este proyecto, recibimos como base una aplicación de administración de conciertos para el Antel Arena. El proyecto constaba de un backend programado en C# y .NET 5.0, el cual incluía una API que se comunicaba con el frontend desarrollado en Angular.
Las issues generadas a lo largo del proyecto no fueron cerradas durante el transcurso, simplemente las fuimos colocando en la columna Done de cada tablero, sin embargo consideramos que deberíamos de haber ido cerrando las issues a medida que se daban por concluidas.
Al final de cada entrega realizamos una retrospectiva utilizando el método DAKI (drop, add, keep, improve) para reflexionar sobre nuestro proceso de trabajo y encontrar áreas en las que pudiésemos mejorar, las que se fueron reflejando en los distintos informes de avance.
A continuación hacemos un breve repaso por cada entrega:

Primera entrega:
En la primera entrega, iniciamos definiendo el marco general de KANBAN para nuestro proceso de ingeniería. También establecimos una guía de uso, versionado y mantenimiento del repositorio en GitHub. Además, llevamos a cabo un análisis de la deuda técnica presente en el proyecto mencionado.
En nuestro proceso de ingeniería hicimos definición de los roles y tomamos la decisión de utilizar dos tableros, uno para la gestión y el otro para llevar a cabo las tareas de desarrollo, esto lo pensamos de esta forma porque consideramos que las tareas de gestión iban a tener un distinto proceso a las de desarrollo, por lo que no iban a tener muchas columnas en común. En el tablero de desarrollo definimos más columnas de las que necesitaríamos en primera instancia, pero luego en las siguientes entregas fuimos modificando el proceso.[Ver proceso de ingeniería]
En cuanto al uso de repositorio elegimos usar la política de trabajo TBD (Trunk based development), realizando cambios e integrándolos rápidamente en la rama develop, en vez de trabajar en ramas separadas. Esta decisión fue para reducir los tiempos de entrega y minimizar conflictos de integración.[Ver guia de repositorio]
Para el análisis de la deuda técnica procedimos a realizar un análisis estático del código mediante la herramienta Sonarqube dado que, de forma gratuita, nos permite analizar y detectar los errores de código automáticamente, además de obtener un estado de la confiabilidad, mantenibilidad y seguridad del proyecto. También analizamos la solución

respecto a su diseño de paquetes observando la estructura en los diagramas de la documentación para poder comprenderla más fácilmente. Por último realizamos un análisis exploratorio del frontend para detectar las funcionalidades que no funcionaban como se esperaba.[Ver análisis de deuda técnica]
En esta entrega se generaron 46 issues, 36 de los cuales corresponden a los bugs y mejoras detectados en el análisis de la deuda técnica, y el resto fueron utilizados en nuestro tablero de gestión, para realizar las tareas correspondiente a administración y documentación de la entrega.[Ver resumen de issues]
En cuanto a las métricas al comienzo no se nota un rendimiento esperado, debido a que no se representan los tiempos utilizados en la gestión, y hubo mucho trabajo de coordinación y puesta en funcionamiento del equipo. De todas formas pudimos observar cuellos de botella en el diagrama de flujo acumulativo en la columna de testing exploratorio, y en el décimo día se detectó un WIP de 4, por lo que un día uno de los integrantes estuvo con dos tareas al mismo tiempo.[Ver métricas][Ver análisis de métricas]
Finalmente en el informe de avance luego de realizar la retrospectiva pudimos identificar varios aspectos para eliminar o reducir en lo referido a la organización y duración de las reuniones, también introducir el uso de una herramienta de cronometraje para medir los tiempos empleados. Como aspectos positivos y a mantener surgieron la buena comunicación y distribución de las tareas. [Ver video de retrospectiva] [Ver informe]

Segunda entrega:
En la segunda entrega, comenzamos a utilizar el tablero KANBAN con el proceso de ingeniería previamente definido, pero con algunas modificaciones. Procedimos a reparar dos de los errores detectados en la primera entrega, seleccionándolos según su gravedad. Asimismo, iniciamos la implementación del pipeline en *GitHub Actions* para automatizar las pruebas del proyecto.
En cuanto al proceso de ingeniería tuvimos que introducir el cambio de omitir la columna de ingeniería de requerimientos, pues en esta entrega no era necesaria ya que los issues habían sido identificados y clasificados en la primera entrega..[Ver proceso de ingeniería]
El pipeline fue configurado para que se ejecute cuando se realiza un *push* o *pull request* en las ramas *main* y *develop*. Este *pipeline* lo configuramos para automatizar los tests unitarios y poder obtener los reportes de las pruebas y el análisis de cobertura. [Ver configuración del pipeline]
De acuerdo a lo solicitado por la rúbrica elegimos reparar los bugs denominados “”Arreglar funcionalidad de Exportación/Importación” y “Arreglar el modificar concierto para mantener artistas”, pues el criterio utilizado fue seleccionar los bugs de mayor prioridad y ante igual prioridad elegir los de mayor severidad de acuerdo a lo establecido por el PO y tester respectivamente. El primer bug se consideró de prioridad alta debido a que no funcionaba y era un requerimiento de la letra, y el segundo bug afectaba la funcionalidad general del sistema. [Ver justificación de bugs]
No se abrieron issues nuevas, pues se utilizaron 2 de las generadas en la entrega anterior, que fueron las correspondientes a los bugs a reparar. En esta ocasión no se generaron issues para el tablero de gestión, sino que se agregaron las tareas al tablero sin crear issues.
En cuanto a las métricas, el *Lead Time* descendió mientras que el *Touch Time* y el *Mean Flow Efficency* aumentaron respecto a la entrega anterior. El *Throughput* se mantuvo constante en 4 unidades, por lo que el equipo logró mantener un nivel de entrega de trabajo consistente en las primeras dos entregas También se volvió a detectar un WIP de 4, por lo

que un día uno de los integrantes estuvo con dos tareas al mismo tiempo.[Ver métricas] [Ver análisis de métricas]
Finalmente en el informe de avance luego de realizar la retrospectiva pudimos identificar varios aspectos para eliminar o reducir como la presencia de columnas cruzadas entre los tableros y la programación de reuniones sin tiempo disponible entre ellas para avanzar en las tareas. También nos propusimos mejorar la gestión del tablero y mantenerlo actualizado. Nuevamente como aspectos positivos y a mantener surgieron la buena comunicación y distribución de las tareas. [Ver video de retrospectiva] [Ver informe]

Tercera entrega:
Para la tercera entrega, se nos solicitó el desarrollo de dos nuevas funcionalidades, lo cual implicó una redefinición de nuestro tablero KANBAN. Además, llevamos a cabo los escenarios de prueba de estas funcionalidades mediante el uso de BDD (*Behavior Driven Development*) con el framework *SpecFlow* en el lenguaje *Gherkin*.
El pipeline no fue modificado en esta entrega, ya que no incluimos la automatización de las pruebas BDD realizadas, debido a la complejidad que se nos presentó para restaurar la base de datos de prueba cada vez que se ejecutaban. [Ver configuración del pipeline]
En cuanto a los roles, la definición se mantuvo y continuamos rotandolos, para que todos los integrantes puedan aprender distintas habilidades en el proyecto. Se continuó implementando dos tableros Kanban, manteniendo incambiado el de gestión anterior y continuamos con la evolución del tablero dedicado al desarrollo. Como agregado a la instancia anterior, en esta nueva, se presentaron 2 requerimientos por lo que el tablero fue ajustado añadiendo algunos procesos. [Ver explicación del tablero] [Ver definición del proceso de ingeniería]
En esta entrega se generaron 8 issues, correspondientes a las acciones de alta, baja y modificación de snacks y a las funcionalidades de ver y comprar snacks, también se agregaron issues separadas para la implementación del front end, todas estas fueron incluidas en nuestro tablero de desarrollo. En esta ocasión no se generaron issues para el tablero de gestión, sino que se agregaron las tareas al tablero sin crear issues.
En el análisis de las métricas vemos que el promedio de *Lead Time* alto indica que el flujo de trabajo en esta iteración no fue ni rápido ni eficiente y que existieron cuellos de botella. El tiempo promedio de *Touch Time* fue ligeramente alto pero aceptable. *El Mean Flow Efficiency* estuvo lejos de cumplir con los objetivos de las heurísticas planteadas en el curso. La eficiencia del flujo promedio se vio afectada debido a que las tareas estuvieron mucho tiempo aguardando a que se incluyese alguna acción en particular (en este caso en *Requirement Definition* y *Test Cases Implementation*). En cuanto al WIP que llegó a un máximo de 7 se explica porque todas las tareas fueron realizadas en simultáneo, lo que a su vez indica un desequilibrio en el flujo de trabajo o una sobrecarga en el equipo.[Ver métricas] [Ver análisis de métricas]
Finalmente en el informe de avance luego de realizar la retrospectiva, entre otras cosas, vimos que debemos solicitar asistencia al grupo docente al momento de estancarnos por desconocimiento de las tecnologías, también nos propusimos identificar los posibles cuellos de botella al principio de la entrega, dándonos tiempo para poder reaccionar con tiempo. Como punto positivo se presentó la situación de que realizamos solicitud de asistencia a tiempo al momento de bloqueo en las tareas. Otra cosa que surgió fue mejorar la forma de documentar la retrospectiva, a efecto de que no sea necesario ver el video para poder entenderla.[Ver video de retrospectiva] [Ver informe]

Cuarta entrega:
En la cuarta entrega, realizamos un análisis exploratorio de los cambios efectuados en la segunda y tercera entrega. Utilizamos la herramienta *Selenium* para automatizar este proceso. También llevamos a cabo un análisis de las métricas obtenidas a lo largo de todo el proyecto, analizando los tiempos de entrega, trabajo en progreso (WIP), tiempos de interacción, entre otros.
Las pruebas fueron implementadas y ejecutadas mediante el uso de la herramienta *Selenium IDE* y cada una corresponde a los distintos escenarios de las tarjetas escritas en formato BDD realizadas para la tercera entrega. Algunos de los escenarios no pudieron ser implementados debido a limitaciones que se nos presentaron para validar áreas de la interfaz que no tenían permiso por parte de usuarios sin los roles correspondientes. Las pruebas no fueron automatizadas en el pipeline de github por dificultades que se nos presentan al tener que hacer un restore de la base de datos de prueba cada vez que se realiza la ejecución de ellas. Sumado a la dificultad para levantar el *backend* y luego indicar al *frontend* el puerto autogenerado correspondiente a la API. [Ver evidencia de pruebas]
Se crearon 7 issues que fueron agregados al tablero KANBAN de desarrollo, y que corresponden a la automatización de las pruebas *frontend* con *Selenium* de los bugs arreglados en la segunda entrega y de las nuevas funcionalidades implementadas en la tercera entrega. Nuevamente no se generaron issues para el tablero de gestión, sino que se agregaron las tareas al tablero.
En cuanto a las métricas, surgió un problema al mantener las tareas en *Review* y no ser finalizadas, por lo que el histograma presenta una sola columna al final lo que indica la pérdida de información detallada de la entrega. El promedio de *Lead Time* representa una mejoría frente a la entrega anterior. El tiempo promedio de *Touch Time* se considera a nuestro entender adecuado y representa que las tareas fueron completadas con la agilidad buscada. El *Mean Flow Efficiency* tuvo un valor apenas adecuado según las heurísticas. Si consideramos el *Touch Time* individual de cada tarea, podemos entender que el flujo hubiese sido mucho mejor si las tareas no se hubiesen mantenido en *Review* ya que el promedio bajó considerablemente producto de tareas que fueron completadas en muy poco tiempo.[Ver métricas] [Ver análisis de métricas]
Finalmente en el informe de avance luego de realizar la retrospectiva, entre otras cosas, se sugirió investigar por nuestra cuenta los temas o tecnologías nuevas al comienzo de cada iteración, de manera que podamos avanzar sin obstáculos. También se volvió a surgir la discusión de mejorar el mantenimiento del tablero e ir registrando los tiempos, asegurándonos de devolver tarjetas que no se estén trabajando para que no queden en el WIP. Se destaca nuevamente la importancia de mantener la comunicación fluida y el apoyo en equipo. Por último, al tratarse de la última entrega hemos expresado nuestro deseo de celebrar los logros que hemos obtenido a lo largo del proyecto. Sabemos de la importancia de reconocer y valorar el progreso y los resultados positivos alcanzados. Por lo tanto, nos gustaría encontrar una manera significativa de conmemorar y compartir estos logros dentro del equipo.[Ver video de retrospectiva] [Ver informe]

Aprendizajes

A continuación presentamos los aprendizajes extraídos de la práctica de este obligatorio, no se incluyen aprendizajes teóricos. Todos los aprendizajes están categorizados por la técnica

utilizada, su título representa el aprendizaje (o confirmación de algo que sabíamos que se debía cumplir y fue realmente observado), y el texto es una breve descripción del por qué del aprendizaje.

Aplicar un marco de gestión ágil

Monitoreo de métricas

Entendemos que la mejor práctica es monitorear las métricas de manera continua durante el desarrollo, nosotros no completamos nuestra tarea de esta forma y pudimos aprenderlo en retrospectiva. Para que sea posible se debe llevar un cálculo actualizado de las métricas y realizar una selección sobre las que analizaremos para tener un panorama de los aspectos relevantes.

Disponer de histórico de métricas

Tan importante como el análisis de las métricas en sí, es disponer de un histórico para su comparación. Un número aislado no sirve de nada si no tenemos algo contra qué compararlo. En este sentido el obligatorio no nos presenta la oportunidad de analizar nuestro trabajo contra una referencia, tal vez sería interesante introducir métricas deseadas en la letra de la propuesta.

Métricas suficientes, no más

El curso presenta varias métricas y existen muchísimas más, debemos ser capaces de seleccionar las métricas que estén alineadas con lo que queremos mejorar y no más que esas, ya que podemos introducir confusión en el análisis. Un ejemplo claro es la duda surgida en clase sobre el cálculo de MTTR para este trabajo, métrica claramente no aplicable a este contexto.

Toma de decisiones informada

Una decisión siempre debe estar basada en datos, en este sentido las métricas proporcionan la información necesaria para identificar áreas de mejora, priorización de tareas y evaluar los cambios introducidos comparando las nuevas métricas obtenidas contra el histórico.

DAKI sobre Aplicar un marco de gestión ágil:

- Drop
    - La necesidad de dependencia docente
    - La falta de celebración de los logros
- Add
    - Herramientas variadas para la retrospectiva
    - Retrospectivas de frecuencia fija y no solamente al finalizar una entrega
    - Análisis de métricas en las retrospectivas
- Keep
    - La motivación del equipo en trabajar con un marco de gestión nuevo
    - El uso de gráficos para las métricas generadas
    - El compromiso del equipo con el marco de gestión y con el proyecto en sí

    - La comunicación de calidad
    - El cumplimiento de los objetivos en los tiempos propuestos
- Improve
    - El ajuste de las técnicas a nuestras necesidades
    - Nivel de detalle de la documentación generada
    - El uso de la revisión del trabajo de un colega sin afectar las métricas

Deuda técnica

Herramientas de apoyo y expansión del concepto

Este trabajo permitió utilizar herramientas para el análisis de código estático (SonarQube) y comprender el resultado del reporte recibido a alto nivel. Además, introdujo la existencia de diversos tipos de deuda técnica asociadas a las distintas etapas del desarrollo (diseño, codificación, pruebas, dependencias y documentación son las observadas principalmente).

Visualización del concepto

Generalmente se considera la deuda técnica como algo abstracto que implica la postergación de tareas. La herramienta utilizada estimaba una aproximación del esfuerzo necesario para pagar la deuda; observar este valor fue interesante ya que nos brinda una idea del tiempo adicional requerido para un proyecto pequeño, y permite entender que si no se maneja correctamente puede escalar a niveles inmanejables dentro de un proyecto de gran porte.

Mitigar la deuda técnica

Una vez identificada, es necesario desarrollar una estrategia para controlar la deuda técnica y corregir, inicialmente, los aspectos prioritarios. El equipo reconoce que no se plantea una planificación ni se aborda nuevamente esta temática más allá de la primera entrega, y entiende que constituye una lección aprendida.

Análisis continuo

Durante el desarrollo de las funcionalidades y la corrección de las incidencias solicitadas, el equipo es consciente de la potencial introducción de más deuda técnica. Este factor determina la necesidad de continuo análisis para mantenerlo bajo los niveles controlables.

DAKI sobre el análisis de deuda técnica:

- Drop
    - Falta de seguimiento en la deuda técnica de las nuevas funcionalidades
    - No trabajar en disminuir la deuda técnica
- Add
    - El uso de más herramientas automáticas
    - Análisis de métricas
    - Un registro histórico de deuda técnica para conocer su evolución
- Keep
    - El stack tecnológico actual
    - La conciencia generada sobre la deuda técnica

- Improve
    - La profundidad del análisis
    - Desarrollo de software introduciendo la menor deuda técnica posible

Implementar un repositorio y procedimientos de versionado

Repositorio

Todos los integrantes del equipo ya habíamos utilizado GitHub como repositorio, pero es interesante comprender que existen diversas funcionalidades de las cuales desconocemos su aplicación.

Flujo de trabajo

La utilización de un flujo de trabajo donde fuese posible integrar los avances rápidamente y disponer de una versión del código siempre funcional nos permitió entender a alto nivel qué es lo que se pretende al desarrollar software colaborativamente de forma eficiente.

Insuficiencia

Creemos que tanto en la utilización de funcionalidades disponibles en GitHub como en el flujo de trabajo el alcance del obligatorio es insuficiente, solamente nos permite entender superficialmente la práctica.

Integración continua utilizando en trunk based

Utilizando este tipo de desarrollo pudimos integrar continuamente el trabajo realizado a la rama de desarrollo lo que nos posibilitó detectar problemas de integración lo suficientemente pequeños para llegar a una resolución del conflicto sencilla.

Conflictos pequeños e infrecuentes

Integrar cambios pequeños y con alta frecuencia impactó en los conflictos haciendo que estos se redujesen en cantidad, tamaño y complejidad de resolución.

Avance visible

El integrar continuamente el trabajo nos permitió visualizar el avance claramente. Además, facilitó la colaboración entre los integrantes del equipo.

Trabajo fácilmente adaptable

El mantener una integración rápida del trabajo nos permitió que pudiésemos redireccionar el esfuerzo si no estábamos encaminados en la solución esperada.

Gestión de las ramas sencilla

Este obligatorio presentó la particularidad de ser 3 integrantes trabajando en simultáneo y no consumir más de 20 minutos por entrega a la gestión de las ramas (resolviendo conflictos por ejemplo). En otras experiencias esta actividad representaba un punto de dolor.

DAKI sobre implementar un repositorio y procedimientos de versionado:

- Drop
    - De usar issues que no son cerradas en su finalización
    - Realizar *merge* sin aguardar que el pipeline complete su ejecución
- Add
    - Proceso de versionado formal, su definición no es parte de este trabajo
    - Uso de *tags*
    - Uso de *releases* para las entregas
    - Uso frecuente de *pull requests* con revisiones de código
- Keep
    - El uso de trunk based para las ramas
    - El uso de una rama *develop* y liberaciones a *main*
- Improve
    - Uso de funcionalidades en forma

Integrar prácticas de QA en el pipeline y gestionar el feedback

GitHub Actions, automatización y personalización con manejo de eventos

Entendemos que se permite automatizar muchos aspectos del desarrollo pero el alcance del curso apenas introduce este concepto de manera práctica. En cuanto a la personalización del flujo de trabajo, fuimos capaces de definir acciones a ser llevadas a cabo en el orden especificado según nuestra conveniencia manejando los eventos en el repositorio (por ejemplo, los merge o los pull request). Nos hubiese gustado disponer de más tiempo para profundizar sobre las distintas opciones que no fueron exploradas. Pese a lo anterior, la herramienta posee un gran potencial por lo que es altamente recomendable y probablemente sea aplicado en el resto de la carrera.

Pipeline, suficiencia limitada

Creemos que la herramienta tuvo un uso suficiente para este trabajo, pero que sería interesante incluir más automatización (por ejemplo las pruebas realizadas con Selenium).

Pipeline, rápida retroalimentación

Implementar el pipeline nos permitió visualizar el estado del código en todo momento, dado que solamente lo utilizamos en una entrega no podemos extraer un aprendizaje más interesante, pero es una herramienta que intentaremos aplicar a proyectos de otras materias.

Pipeline, confiabilidad

El utilizar el pipeline nos permitió confiar en que el código que estábamos subiendo al repositorio no alteraba el trabajo de un compañero aunque no supiéramos exactamente los detalles de los cambios realizados por esa persona.

DAKI sobre integrar prácticas de QA en el pipeline y gestionar el feedback
:

- Drop
    - Eliminar los procesos manuales restantes
- Add
    - Ejecución de pruebas en Selenium
    - Análisis estático de código en el pipeline
    - Ejecución de pruebas eliminando rutas escritas a fuego
    - Más automatización al pipeline
- Keep
    - La investigación en tecnologías y funcionalidades integrables
    - Reportes generados automáticamente
- Improve
    - Optimizar los tiempos de ejecución de las pruebas

Generar escenarios de testing desde la perspectiva del usuario

BDD, las tarjetas US (User story) son la base

Al definir tarjetas para las US con CCC se definen escenarios que representan los criterios de aceptación y comportamientos esperados de la funcionalidad constituyendo un punto de partida para el desarrollo y el testing.

BDD, documentación con código

Cada una de las pruebas realizadas a partir de los escenarios de BDD pueden ser considerados como documentación clara del comportamiento de cada funcionalidad.

BDD, punto intermedio entre el negocio y lo técnico

Al tener un requerimiento y elaborar una US que presenta escenarios y criterios de aceptación, la tarjeta en sí misma es fácilmente verificable con un cliente y representa el punto de partida de testing y desarrollo. Por esto, podemos considerarla un punto intermedio entre dos áreas que potencialmente pueden introducir desvíos entre lo esperado y lo desarrollado.

DAKI sobre generar escenarios de testing desde la perspectiva del usuario

- Drop
    - Casos que no reflejan uso real del sistema
- Add
    - Casos de prueba en a través de distintos dispositivos y navegadores
    - Casos de prueba con diferentes perfiles de usuario
- Keep
    - Escenarios de prueba que cubran distintos perfiles de usuario
    - Escenarios de prueba para funcionalidades considerando datos correctos e incorrectos
    - Descripción detallada de los escenarios de prueba

- Improve
    - Asegurar la cobertura de las funcionalidades con usuarios habilitados y no habilitados a realizar una funcionalidad

Automatizar el testing funcional o de caja negra

Automatización

Al utilizar Selenium conseguimos automatizar los test realizados al front end y entender las bases de una herramienta que permite ahorrar mucho tiempo en pruebas que generalmente son ejecutadas de forma manual.

Varios navegadores, mismo esfuerzo

Ejecutar la misma *Test suite* de pruebas en varios navegadores implica poco esfuerzo adicional adaptando los casos de prueba de los que se dispone. Quizás al tener más experiencia con la herramienta este esfuerzo puede reducirse aún más.

Utilidad ante una regresión

Selenium permite automatizar test de caja negra realizados a través del front end lo que entendemos es de mucha utilidad para pruebas de regresiones, especialmente en productos de grandes dimensiones.

DAKI sobre automatizar el testing funcional o de caja negra

- Drop
    - la ejecución manual de todo caso de prueba automatizable
    - pasos innecesarios en los casos de prueba
- Add
    - Reestablecer la base de datos para pruebas automáticamente
    - Ejecutar automáticamente las pruebas en el pipeline
    - Situaciones de carga del sistema
- Keep
    - La generación de los casos de prueba que estén alineados con las US
    - Separación de los casos de prueba en *Test Suites*
- Improve
    - La ejecución no es muy estable (a veces se generan casos de falla por demoras o problemas en los datos)
    - Lograr ejecuciones de los casos de prueba más rápidas

Reflexionar sobre DevOps

Buena comunicación y Equipo colaborativo

En la realización de este obligatorio, todos los integrantes del equipo estamos de acuerdo que como resultado de las técnicas aplicadas la comunicación fue constante y de calidad, al

mismo tiempo que se potenció la colaboración del equipo ayudándonos cuando alguien experimentaba una dificultad.

Automatización

La mayor visibilidad en cuanto al trabajo realizado es la automatización del pipeline, generando confianza en el desarrollo, un monitoreo constante del estado de la aplicación y un producto potencialmente entregable en todo momento.

Retroalimentación rápida

Un aspecto importante sobre DevOps es la rápida retroalimentación que se obtiene al ejecutar las pruebas cuando se dispara uno de los eventos del pipeline. Esto puede ser potenciado al incluir más acciones en un proyecto de mayor porte.

DAKI sobre DevOps

- Drop
  - Rutas harcodeadas en la solución
- Add
  - Correr pruebas de Selenium cuando se dispara un evento en el pipeline
  - Incorporar contenedores (nos hubiese gustado hacerlo)
  - Incorporar herramientas de monitoreo para identificar cuellos de botella
- Keep
  - Exploración de la tecnología asociada
  - Comunicación y colaboración del equipo
  - Mejorar y aprender continuamente
  - Aplicación de DevOps
- Improve
  - Incrementar la automatización
  - La gestión de la revisión de código

Lecciones Aprendidas

Las lecciones aprendidas de las entregas 1 a la 4 fueron transcritas al formato presentado en clase y se pueden encontrar en el siguiente enlace. Siguen a continuación las lecciones aprendidas para el informe académico:

Formato de lecciones aprendidas
Si se deben escribir lecciones aprendidas se debe utilizar el formato *Título. si <condición> entonces <sugerencia>. Anécdota <2 líneas>.*
El equipo desconocía esta estructura y generaba lecciones aprendidas que eran un párrafo (muy largas).

Acortar el tiempo de la segunda vía

Si se desea mejorar los procesos del trabajo realizado y no cometer los mismos errores, se debe acortar el tiempo transcurrido hasta que se recibe la retroalimentación.
El equipo recibió retroalimentación docente para la tercera entrega por lo que se aprecia que muchos comentarios fueron incluídos en a partir de entonces. Al no disponer de otra retroalimentación no pudimos acercarnos más a lo esperado por la cátedra.

Simplificar el flujo de trabajo
Si el tablero Kanban se vuelve demasiado grande, complejo y/o desordenado, entonces es recomendable simplificar la estructura y limitar la cantidad de columnas.
Al utilizar dos tableros kanban (gestión y desarrollo) logramos reducir las columnas en nuestro tablero Kanban principal (desarrollo), permitiendo visualizar y gestionar mejor nuestras tareas, lo que condujo a una mayor eficiencia en el flujo de trabajo en ambos.

Establecer límites para el WIP
Si existen problemas de sobrecarga y/o multitarea, se debe establecer límites para el WIP en cada columna del tablero evitando acumulación de tareas.
El equipo entiende que acumular muchas tareas en la columna de revisión fue perjudicial tanto para la sobrecarga como para la afectación de las métricas calculadas.

Transparencia y colaboración de trabajo
Si se quiere compartir el conocimiento se debe generar colaboración, transparencia en las actividades realizadas y comunicación entre los integrantes del equipo.
Al siempre saber qué estaba haciendo un compañero, cómo lo estaba haciendo y ayudarnos cuando nos bloqueamos conseguimos aprender de los aciertos y errores grupales.

Evitar el exceso de gestión
Si se sobrecarga el tablero con actividades que no son fundamentales se puede perder el foco en lo prioritario y genera sobrecarga en la gestión.
El equipo comenzó utilizando el tablero de gestión minuciosamente generando tareas para actividades como “entregar en aulas” que no aportaban valor al producto entregado y generaban sobrecarga al mantenimiento del mismo.

Establecer criterios de aceptación claros
Si se desea saber si un producto cumple con lo deseado se deben establecer criterios de aceptación claros y compartirlos.
Parte de la retroalimentación docente es establecer criterios de aceptación claros y el equipo reconoce que pudo haberlo realizado explícitamente. Constituye una lección aprendida general.

Adaptar kanban a las necesidades del equipo
Si se desea obtener mejores resultados se debe adaptar la metodología a las necesidades del equipo generando un punto intermedio entre la teoría y la práctica.
El principal ajuste generado por el equipo fue la utilización de dos tableros y la realización de una retrospectiva para analizar el trabajo de la entrega y no semanalmente como es recomendado.

Establecer expectativas claras de entrega

Si se quiere obtener un producto dentro de los parámetros deseados se debe establecer expectativas claras de la entrega.
Entendemos que en muchas ocasiones fue necesario compartir en clase un momento de discusión para comprender las expectativas de la entrega, muchos de esos momentos fueron iniciados por un integrante de este equipo.

Cerrar Issues cuando son finalizadas
Si utiliza *issues* en GitHub, debe cerrarlas conforme son completadas
En este trabajo reconocemos que pocas veces cerramos issues, realmente no nos acordamos de realizarlo en la mayoría de las oportunidades.

Trabajar sobre la deuda técnica
Si se realiza un análisis de deuda técnica debe realizarse un plan para su corrección.
El equipo dedicó varias horas al análisis de deuda técnica pero desconocía la necesidad de elaborar un plan para su corrección, mencionado en la devolución docente

Conclusiones

A lo largo de la duración de este proyecto, nuestro equipo ha experimentado y aplicado varias técnicas y metodologías de la ingeniería de software ágil. En este documento, sintetizamos y reflexionamos sobre lo realizado y aprendido en el transcurso del mismo.

Nuestros esfuerzos iniciales se centraron en probar, explorar y analizar el código existente proporcionado, identificando y clasificando varios errores y analizando la deuda técnica. Este proceso resultó crucial para entender la calidad del código inicial y trazar un plan de trabajo para abordar los problemas encontrados. Aprendimos que la organización del trabajo es esencial y que las herramientas como Kanban son valiosas para mantener el flujo de trabajo visual y gestionable.

En la segunda etapa, resolvimos algunos de los errores identificados utilizando *Test Driven Development (TDD)*. Esta metodología fue beneficiosa en prevenir la reintroducción de errores y en mantener la funcionalidad existente intacta mientras se avanzaba en las mejoras. Adicionalmente, la implementación de un pipeline con *GitHub Actions* para ejecutar pruebas automáticas y realizar análisis de seguridad fue crucial para mejorar la calidad del código y la eficiencia del equipo.

Para el tercer hito del proyecto, nos enfrentamos al reto de incorporar nuevas funcionalidades utilizando *Behaviour Driven Development (BDD)* con la ayuda de *SpecFlow*. Esta experiencia reafirmó la importancia de las pruebas en el desarrollo de software, y cómo BDD puede facilitar la comprensión de los requisitos y fomentar la colaboración entre los roles técnicos y no técnicos del equipo.

En la cuarta etapa, automatizamos pruebas exploratorias con *Selenium* y calculamos métricas de Kanban. Estas acciones nos permitieron detectar y resolver posibles cuellos de

botella y optimizar aún más nuestro proceso de trabajo, consiguiendo observar en retrospectiva nuestro desempeño.

Las implicaciones de las lecciones aprendidas durante este proyecto son significativas. Hemos comprendido la importancia de las pruebas, tanto exploratorias como automatizadas, y cómo éstas pueden mejorar la calidad del producto y la eficiencia del equipo. Además, hemos experimentado el valor de las metodologías ágiles como TDD y BDD, así como de las herramientas de automatización, como *GitHub Actions* y *Selenium*.

Estas conclusiones son importantes porque demuestran que un enfoque disciplinado y metódico puede llevar a resultados significativos en la calidad del software y en la eficiencia del equipo. Además, sugieren que la inversión de tiempo en pruebas y automatización no sólo es beneficiosa, sino necesaria en el desarrollo de software ágil.

En cuanto a lo que estas conclusiones implican para el futuro, nos indican la dirección a seguir. Nuestro equipo se ha convertido en un grupo más efectivo y cohesivo, y estos aprendizajes servirán de guía para futuros proyectos. La importancia de las pruebas, la automatización y las metodologías ágiles se ha destacado a lo largo de este proyecto, y continuaremos utilizando y perfeccionando estas prácticas en el futuro. Además, el análisis regular de las métricas del equipo nos permitirá mantener un alto nivel de rendimiento y realizar ajustes en el momento en que sea necesario.

Como conclusión final, este proyecto ha sido una experiencia valiosa que nos ha permitido aplicar y expandir nuestros conocimientos de ingeniería de software ágil. Con la experiencia ganada y las lecciones aprendidas, nos sentimos motivados para enfrentar los retos futuros que nos esperan en el campo de la ingeniería de software.

Guia de instalación para el desarrollo y despliegue en producción

El documento puede encontrarse siguiendo el siguiente enlace.