Universidad ORT

Proyecto DevOps

Ingeniería de Software Ágil 2

Romina Giaccio - 206127
Sofia Begino - 230763
Joaquin Sommer - 184441

Repositorio: https://github.com/sofiabegino/isa2-230763-206127-184441.git

02/06/2023

ÍNDICE

Resumen de gestión del proyecto....2
Reflexiones sobre el aprendizaje.... 5
Aplicar un marco de gestión ágil.... 5
Analizar la deuda técnica.... 5
Implementar un repositorio y procedimientos de versionado....5
Crear un pipeline con eventos y acciones....6
Integrar prácticas de QA en el pipeline y gestionar el feedback.... 6
Generar escenarios de testing desde la perspectiva del usuario....6
Automatizar el testing funcional o de caja negra....6
Reflexionar sobre DevOps.... 7
Lecciones aprendidas.... 8
Lecciones aprendidas - Entrega 1....8
Lecciones aprendidas -Entrega 2....8
Lecciones aprendidas - Entrega 3....9
Lecciones aprendidas - Entrega 4....9
Conclusiones....11
Guía de instalación para desarrollo y despliegue en producción....12
Instalación de la app y ejecución de la misma.... 12
Ejecución de tests unitarios....12
Ejecución de tests funcionales.... 13

Resumen de gestión del proyecto

Para la gestión del obligatorio decidimos ubicar toda la documentación dentro de una carpeta “Entregas”, donde creamos una carpeta para cada etapa.

La **etapa 1** consistió principalmente en la definición del marco general de KANBAN, una primera versión del proceso de ingeniería, la creación del repositorio y definición de roles. Además, realizar un análisis de deuda técnica.

Comenzamos creando el repositorio en github y asociando un project board, lugar donde se encuentra nuestro tablero de tareas. Definimos las columnas que consideramos necesarias: “Backlog”, “In Progress”, “Testing” y “Done”. Luego, cargamos los issues necesarios para cumplir con los requerimientos de la entrega. Estipulamos el uso de gitflow como estrategia de branching. Hicimos el análisis de deuda técnica, donde se analizaron tanto backend como frontend, esto nos permitió ganar un mayor entendimiento sobre la aplicación y los requerimientos funcionales que comprende. Además, creamos los templates para los issues. Finalmente, realizamos una retrospectiva usando el método DAKI.

Aquí no definimos el esfuerzo estimado de las tareas, pero al finalizar indicamos el esfuerzo real en horas-persona por cada tarea, donde vimos que el total de la iteración fue de 30.5 horas-persona. Medimos las ‘horas-persona’ por tarea como las horas totales que se dedicaron a una tarea por parte de todo el equipo; si más de una persona trabajo en una misma tarea, se suman las horas de cada integrante.

La **etapa 2** consistió de la creación de la primera versión del tablero en función del proceso de ingeniería enmarcado en KABNAN y la configuración del pipeline en función del tablero antes mencionado. Elección y reparación (usando TDD) de dos bugs de mayor severidad. Por último, realizar una review de los bugs con el PO y una retrospectiva basada en el método DAKI.

Comenzamos realizando algunas adaptaciones respecto al proceso de ingeniería presentado en la primera entrega. Ahora, incorporamos la estrategia de BDD. Nuestras columnas fueron las siguientes: “Backlog”, “In Progress”, “Requirements definition”, “Application implementation”, “Testing”, “Refactoring” y “Done”. El flujo a seguir depende del tipo de la card (issues de documentación vs issues de bugs).
Luego, elegimos dos de los bugs con mayor severidad: “Cambio de contraseña no valida la seguridad” (backend, resuelto con TDD) y “Usuario con rol ‘Artista’ no puede desloguearse” (frontend). Finalmente, realizamos una review de los bugs con el Product Owner y una retrospectiva usando el método DAKI.

En esta instancia estimamos el esfuerzo de las tareas previamente, lo que nos permitió medir qué tan certeros estaban siendo nuestros estimados. Tuvimos un esfuerzo estimado total de 20,5 h-p y un esfuerzo real de 17,9 h-p.

La **etapa 3** consistió en actualizar el proceso de ingeniería basado en BDD, enmarcado en KANBAN, crear la segunda versión del tablero en función de este y finalmente configurar el pipeline en función del tablero. Por otro lado, había que los escenarios de prueba en BDD

para las nuevas funcionalidades y desarrollar el frontend y luego el backend de las mismas. Por último, realizar una review de las nuevas funcionalidades con el PO y una retrospectiva basada en el método DAKI.

Para esta entrega, actualizamos el proceso de ingeniería y adaptamos el tablero en función del mismo. Las columnas resultaron: “Backlog”, “In Progress”, “Requirements definition”, “Test cases implementation”, “Application implementation”, “Testing”, “Refactoring” y “Done”. Luego, desarrollamos las nuevas funcionalidades “mantenimiento de snacks” y “compra de snacks” haciendo uso de BDD. Debido a esto, modificamos el pipeline en github para que corra los tests que se agregaron al usar BDD. Finalmente, realizamos una review de las nuevas funcionalidades con el Product Owner y una retrospectiva usando el método DAKI.

En esta ocasión, el esfuerzo total estimado fue de 34,25 h-persona y es esfuerzo total real fue de 44.5 h-p. En esta ocasión nos llevo más tiempo del estimado, lo que nos hizo dedicar más tiempo a la etapa de estimación de la siguiente entrega.
Además, medimos el esfuerzo por tipo de tarea, donde obtuvimos una imagen más ordenada de la relación proporcional entre las diferentes categorías.

La **etapa 4** consistió en la automatización del test exploratorio de ambos bugs y ambas nuevas funcionalidades usando Selenium. La confección de análisis de métricas DevOps y la realización de una retrospectiva basada en el método DAKI.

Previamente, en el template de issues de User Story teníamos el esfuerzo dividido en las etapas de “Test Cases Implementation”, “Application Implementation”, “Testing” y “Refactoring”. Dado que ya no estamos desarrollando y solo hubo user stories destinadas a la Automización de Tests con Selenium, lo adaptamos a
Estimado: [Estimación en horas-persona]
Real: [Estimación en horas-persona]
Por este mismo motivo, adaptamos nuestro proceso de ingeniería y nuestro tablero. Las columnas resultantes fueron: “Backlog”, “In Progress”, “Requirements definition”, “Testing” y “Done”.
Luego, para la realización de los test funcionales utilizamos Selenium, una herramienta que permite automatizar las pruebas de una aplicación web. Específicamente, utilizamos Selenium IDE, una extensión de Chrome que permite grabar las acciones que se realizan en el navegador y luego reproducirlas. Luego de grabar las acciones básicas se agregaron asserts para verificar que el sistema se comportará como se esperaba.
También nos focalizamos en corregir todo el feedback que nos habían dado de la entrega 1, fuimos punto por punto tomando cada sugerencia.

En esta entrega, hicimos la confección de análisis de las métricas de Kanban: cycle time, lead time, touch time y registro de esfuerzos. También hicimos una pie chart para detallar los esfuerzos por tipo de tarea.
Las métricas cycle time, lead time y touch time las calculamos en días, y los otros dos esfuerzos los calculamos en horas-persona. Los resultados dieron:
Cycle time: 1,1
Lead time: 3,4
Touch time: 1,1
Esfuerzo total en horas persona: 21

Esfuerzo por tipo de tarea: corrección de feedback (17%), documentación (35,4%), reuniones (10,4%) y automatización de tests (37,5%).

Para finalizar:
Vale acotar que en cada entrega se realizó la documentación correspondiente a cada tarea realizada, incluyendo los documentos de lecciones aprendidas y de avance de etapa.

Tomamos una issue como cerrada al pasarla a la columna “Done”. Tenemos 79 issues en la columna “Done” y 18 en el backlog que refieren a los bugs reportados luego del análisis de deuda técnica de la entrega 1.

Reflexiones sobre el aprendizaje

Utilizamos el método DAKI para reflexionar sobre cada objetivo. En algunos objetivos no tenemos comentarios para todos las etapas de DAKI(Drop, Add, Keep, Improve). En estos casos simplemente no lo agregamos.

Aplicar un marco de gestión ágil

Drop
Las standups no las pudimos implementar de la mejor manera debido a las características del proyecto. En algunas ocasiones nos pasó que al “terminar” la standup y hablar de los blockers de cada uno y lo que habíamos hecho, continuamos avanzando en otras tareas luego de la reunión, por lo que la línea quedaba un poco difusa.
Keep
La utilización de un tablero ágil fue muy útil para gestionar el proyecto y ver el estado actual de cada tarea.
Improve
En las retrospectivas nos centramos mucho en los problemas de gestión y dejamos de lado los problemas técnicos o de desarrollo que seguramente también teníamos.
El rol de scrum master tampoco tuvo mucho impacto dentro del proyecto. Guiaba la retrospectiva pero fuera de eso no guió al grupo para aplicar mejor la gestión ágil.

Analizar la deuda técnica

Add
Se podría haber utilizado alguna herramienta para el análisis estático del proyecto. Como por ejemplo EsLint.
Keep
Se hizo un muy buen análisis dinámico manualmente donde se encontraron muchos bugs.
Improve
Podemos haber mejorado como estimamos la prioridad de cada bug que se encontraba. Cuando se intentó elegir

Implementar un repositorio y procedimientos de versionado

Drop
En este punto consideramos que no se realizó ningún procedimiento que necesitamos dejar de hacer.
Add
Lo único que podríamos agregar en este punto era intentar aplicar trunk based development donde luego de cada tarea se mergeara a main los cambios.
Keep
Tener solo una rama por tarea/ticket fue muy útil y fácil de trackear para ver el seguimiento de cada tarea.
Improve
Se intentó utilizar un estándar para el nombramiento de los commits pero no siempre se siguió. Podríamos haber sido más rigurosos en aplicar siempre el estándar que definimos en la entrega 1.

El estándar para el nombramiento de las ramas se siguió pero terminó siendo un poco confuso cuando se trataba de ramas de documentación.

Crear un pipeline con eventos y acciones

Add
Hubiese estado bueno agregar el deploy en las actions porque es algo que casi siempre esta y está bueno saber como se hace, pero para este proyecto estaba fuera de scope.
Improve
A parte del build y los tests, podríamos haber agregado algún chequeo de la calidad del código estático. Por ejemplo, poner que se corra un linter.

Integrar prácticas de QA en el pipeline y gestionar el feedback

Keep
Las prácticas de QA nos permitieron disminuir el riesgo de generar bugs a la hora de crear nuevas funcionalidades al sistema. Nos permitió también testear las funcionalidades de distinta manera.

Generar escenarios de testing desde la perspectiva del usuario

Para este punto se va a tener en cuenta los tests que se hicieron utilizando TDD(entrega 2) y BDD(entrega 3).

Keep
En los tests de BDD se agregaron todos los casos posibles que pudimos pensar que podrían pasar desde la perspectiva del usuario. La forma que aplicamos BDD fue muy correcta.
Improve
Para la entrega 2 se podrían haber hecho más casos para cubrir los casos bordes que podrían haber ocurrido.
En el caso de la entrega 3, cuando se hicieron los tests de BDD, se utilizó un mock para simular las respuestas de la base de datos. En un caso ideal creo que no se tendría que haber utilizado esto para también probar la integración con la base de datos.

Automatizar el testing funcional o de caja negra

Add
Podríamos haber buscado la opción de agregar los tests con Selenium al pipeline de github. También se podría haber configurado en alguna página para que estos tests se corran cada cierto tiempo de manera de reportar si hay algún fallo en el flujo del test.
Keep
Hicimos todos los casos bordes que se nos ocurrieron desde la perspectiva del usuario.
Improve
Los tests utilizando Selenium IDE tuvieron ciertas restricciones que se hubieran solucionado si hubiésemos tenido tiempo de implementar algún script personalizado.

Reflexionar sobre DevOps

Add

Establecer un entorno de integración y despliegue continuo para garantizar que el código se integre, pruebe y despliegue de manera eficiente y automática.

Keep

Continuar aplicando prácticas de gestión ágil y QA en el proyecto para garantizar la calidad del software y la adaptabilidad a los cambios.

Lecciones aprendidas

Lecciones aprendidas - Entrega 1

Al principio no teníamos claro cómo organizarnos para realizar las tareas/issues, ni que representar con exactitud en nuestro proyecto y tablero, entonces decidimos tomar las issues como cualquier tarea a realizar y definir templates para las diferentes issues que nos ayudarán a identificarlas. En nuestro caso, para cada entrega, realizamos un análisis previo de cada actividad y artefacto a utilizar o presentar en la entrega (cada punto de la currícula del proyecto *DevOps* en la letra del obligatorio), identificando todas las tareas de documentación, implementación, análisis etc que se requerían, y mapeandolas en issues de al tablero, donde para cada tipo de issue creamos templates diferentes y labels para identificarlas mejor.

Si se espera mantener un orden del proyecto en un repositorio y ubicar rápidamente cada documento del mismo se puede mantener una estructura de carpetas representativa. En nuestro caso, organizamos la estructura de carpetas en el repo en dos niveles, el primer nivel en el que se puede identificar fácilmente cada entrega, ya que se tiene una carpeta destinada a cada una, y un subnivel de carpetas dentro de cada entrega, las cuales fueron nombradas de forma que representen y contengan los documentos a entregar durante cada instancia de entrega.

Lecciones aprendidas -Entrega 2

Si se desea mantener un *control de versiones* entonces se debe tener en cuenta que el mismo podría variar en el tiempo y requerir ciertas modificaciones, por lo que es importante no solo basar las estrategias de branching en estándares, sino que diseñar teniendo en cuenta que las posibles modificaciones que requieran para el proyecto en particular puedan adaptarse e incluirse de forma natural y rápida a nuestro estándar. En nuestro caso, habíamos establecido inicialmente un estándar de branches al que luego debimos modificar para diferenciar ramas de documentación, corrección de errores (fix), bugs y features, además agregando en cada una la descripción corta de la issue que se trabajaba en la branch, cuando inicialmente solo se indicaba el número de issue y el tipo de branch.

Si se desea aplicar un marco de gestión ágil como *Kanban* entonces se deberán realizar dos ceremonias básicas, retrospectivas y Stand Up. Las retrospectivas luego de cada entrega y las stand ups que son diarias, encontramos que en caso de no poder realizar estas ceremonias como indicamos antes lo mejor es hacer una modificación que nos permita ceñirnos lo más posible a la definición original. En nuestro caso nuestro equipo no contaba con la disponibilidad para realizar stand up todos los días, por lo que esta ceremonia se realizaba semanalmente y se intentaba realizar varias veces en la semana de ser posible, además en cada reunión que realizaba el equipo, sin importar el objetivo de la misma (programación en pares, implementación, investigación de una tecnología, etc) tomábamos un momento al inicio de la misma para hablar sobre el estado de las tareas.

Si se desea utilizar tableros Kanban en Github, entonces se puede generar un proyecto en la solapa “Projects” del repositorio, y luego implementar un tablero Kanban utilizando un layout de tipo “tabla”. Dependiendo del tipo de tablero a utilizar y el proceso de ingeniería, se definirán columnas y issues para las diferentes tareas a realizar, no existe una única forma de organizar el tablero, se debe tener en cuenta cómo se ajusta mejor a las necesidades del proyecto y el equipo, además es muy práctico vincular las issues con los pull request generados. En nuestro caso, definimos un tablero horizontal del tipo “*Sustentable Kanban*”, donde pudimos vincular las issues con los pull request generados.

Lecciones aprendidas - Entrega 3

Para generar pruebas *Behavior Driven Development* (BDD) entonces podemos utilizar *SpecFlow*, el cual es un marco de prueba de código abierto para aplicaciones .NET. En nuestro caso como la aplicación proporcionada por los docentes estaba implementada en .NET y contábamos con el requerimiento de implementar una serie de *endpoints* a ser probados mediante pruebas BDD esta fue la herramienta seleccionada para realizar el trabajo, ya que además ayuda a crear pruebas automatizadas escalables y de alta calidad a partir de escenarios de *Gherkin.*

Para realizar entregas más rápidamente en tiempos reducidos entonces es fundamental aplicar el trabajo en conjunto como indican las prácticas *DevOps*. En nuestro caso para esta entrega, notamos que el trabajo en equipo fue fundamental y que en lugar de castigar el error, al notar que no pudimos cumplir todos los puntos debido a problemas de coordinación, decidimos tomar este fracaso para salir adelante en la 4ta entrega, ver qué hicimos mal y mejorarlo, logrando cumplir con todos los puntos solicitados y tratando de abarcar todos los puntos del feedback recibido anteriormente.

En caso de querer compartir un proyecto en un repositorio en *GitHub* que contenga un tablero en Projects con un tercero, entonces se debe tener en cuenta que si el proyecto contiene un tablero, debe darle visualización al tercero sobre el mismo, de lo contrario el usuario podrá ver todo el contenido del repositorio menos el tablero en cuestión. En nuestro caso los docentes podían ver el tablero *Kanban* mediante imágenes en los diferentes documentos del proyecto, pero no podrían acceder al mismo ni ver que se encontraba creado, para visualizar los detalles y el estado de este, ya que no contaban con acceso.

Lecciones aprendidas - Entrega 4

Si se quiere definir un tablero de tipo “*Sustentable Kanban*” acorde al proyecto, es necesario previamente realizar la definición del proceso de ingeniería, identificando qué, quién, cómo y qué se obtiene de cada paso del mismo donde los pasos pueden ir desde Requirement Definition (CCC), Test Cases Implementation, Application Implementation, Testing hasta Refactoring, entre otros. También se requiere definir un proceso de gestión del proyecto, del cual algunos pasos podrían reflejarse también en el tablero. Además considerar que el tablero incluye actividades propias como Backlog y Done. En nuestro caso el proceso de

ingeniería sufrió varias modificaciones al transcurrir las entregas, por lo que estas se debieron reflejar en el tablero de Kanban.

En caso de requerir utilizar una nueva tecnología entonces es una buena idea realizar una investigación sobre la misma, mucho mejor si se puede realizar en conjunto con integrantes del equipo y ver ejemplos de la misma y su funcionamiento. En nuestro caso realizar esta actividad en conjunto con el equipo nos facilitó el uso de Selenium y un mayor entendimiento del mismo incluyendo sus limitaciones.

Si se desea optimizar el tiempo para realizar entregas del proyecto entonces es buena idea dedicar tiempo a estimar las tareas/issues que se requieren para la misma y empezar lo antes posible a realizarlas. Para esto optamos por realizar la estimación de las issues y la asignación de las tareas lo antes posible, ya que si bien nos encontramos en un marco Kanban, existían entregas pautadas y por lo tanto se requería que las tareas/issues fueran estimadas para ser realizadas en un tiempo coherente dentro del plazo de entrega.

Si se necesitan realizar test funcionales de una forma rápida y que puedan ser ejecutados de una forma visual y automática entonces utilizar el IDE de *Selenium* es una buena opción. Durante nuestro proyecto notamos que realizar los test en Selenium no solo resultaba práctico sino que es una herramienta intuitiva y fácil de manejar, en la cual podemos observar la ejecución de los test de nuestra UI para identificar si el flujo que se desea probar esta funcionando correctamente.

Conclusiones

Los resultados de la investigación de la deuda técnica nos permitieron detectar muchos bugs que clasificamos con diferentes niveles de severidad. Así pudimos seleccionar los más críticos y solucionar algunos de los mismos reduciendo parte de la deuda. Esto significa que se han abordado y corregido las deficiencias y problemas técnicos existentes en la aplicación web para estos bugs, que luego de los ajustes realizados, cumplieron con los criterios de aceptación impuestos.

Debido a la cantidad de bugs encontrados consideramos que la deuda técnica es alta, esto puede derivar en diversas consecuencias negativas para una organización o cliente, como por ejemplo: retraso en entregas, mayor costo de mantenimiento, baja calidad de software, entre otros. Al saldar parte de la deuda técnica contribuimos a una mayor estabilidad y confiabilidad del sistema. Al abordar problemas subyacentes, se reduce la posibilidad de errores y fallas, lo que a su vez mejora la experiencia del usuario y aumenta la satisfacción del cliente.

Aparte de esto, se realizaron investigaciones sobre diferentes tecnologías y prácticas ágiles cómo: Kanban, Github para el repositorio y procedimiento de versionado, SpecFlow para la implementación de BDD, Selenium para testing funcional.

Trabajar con el marco de gestión ágil Kanban nos ayudó a mantener un flujo constante de trabajo, identificar dificultades y mejorar la eficiencia del proceso de desarrollo y entrega de software. Github nos permitió trabajar de forma concurrente en el mismo repositorio, gestionar ramas de desarrollo.

SpecFlow nos permitio escribir escenarios de prueba en código Gherkin y luego automatizar estas pruebas utilizando código. Esta técnica de BDD se alinea bien con DevOps, ya que fomenta una comunicación clara y una colaboración efectiva entre los equipos de desarrollo y negocio. Mientras que Selenium nos permitio automatizar las pruebas de interfaz de usuario (UI) en diferentes navegadores y plataformas, lo que facilitó la validación de la funcionalidad del software, ya que las nuevas funcionalidades del software se probaron de manera exhaustiva y consistente en el pipeline de desarrollo.

Por último, la confección de las métricas de Kanban nos resultaron muy útiles. A través de la implementación y seguimiento de estas métricas, hemos podido obtener una visión clara y objetiva del rendimiento de nuestro flujo de trabajo.

Guía de instalación para desarrollo y despliegue en producción

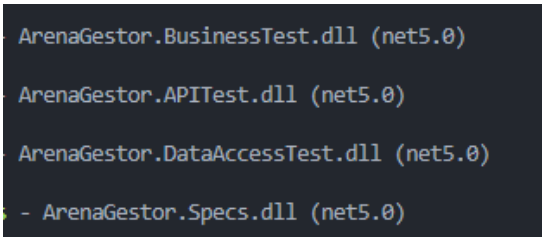
Guía de instalación
Esta guía es para correr la aplicación en el ambiente local. Algunos de los pasos están disponibles como ejemplo en el pipeline de GitHub Actions.

Instalación de la app y ejecución de la misma

1. Instalar dotnet 5.0
2. Instalar node 18.0 en adelante
3. Utilizando SQL Server Management Studio, ejecutar el script "SQL con ConDatos.sql" o recuperar la base desde "BAK con ArenaGestorConDatos.bak" que se encuentra en la carpeta **Obligatorio\Base de datos**.
4. En una consola, estando parado en la carpeta **"Obligatorio\codigo\ArenaGestor"**, correr el comando "dotnet restore" para restaurar los paquetes de nuget
5. En una consola, estando parado en la carpeta **"Obligatorio\codigo\ArenaGestor\ArenaGestor.API"**, correr el comando "dotnet run" para correr el backend
6. En una consola, estando parado en la carpeta **"Obligatorio\codigo\ArenaGestorFront"**, correr el comando "npm install" para restaurar los paquetes de npm
7. En una consola, estando parado en la carpeta **"Obligatorio\codigo\ArenaGestorFront"**, correr el comando "npm start" para correr el frontend
8. Ahora al ir a la url **http://localhost:4200/** se podrá ver la aplicación corriendo en el ambiente local
9. Si hay problemas con la conexión a la base de datos seguramente tengas que cambiar el **ConnectionStrings** en los archivos "**Obligatorio/codigo/ArenaGestor/appsettings.json**" o "**Obligatorio/codigo/ArenaGestor/ArenaGestor.API/appsettings.json**"

Ejecución de tests unitarios

1. En una consola, estando parado en la carpeta **"Obligatorio\codigo\ArenaGestor\ArenaGestor.API"**, correr el comando "dotnet build"
2. Luego correr el comando "dotnet test" para correr los tests. Estas son las cuatro carpetas de tests:



Ejecución de tests funcionales

1. Se tiene que tener la base con los datos ya restaurada.
2. Abrir Selenium IDE, seleccionar "Open an existing project" y seleccionar el archivo "Snack.side" en la carpeta "**Entregas\Entrega4\Selenium Tests\SeleniumProyectos**"
3. Hacer click en "Run all tests"
4. En esa carpeta tambien se encuentran dos proyectos mas donde hay tests para los bugs resueltos en la entrega 2. Hacer los mismos pasos que 1, 2, y 3