Universidad ORT Uruguay

Facultad de Ingeniería

Ingeniería de software ágil 2

Documentación de la entrega 2

https://github.com/IngSoft-ISA2-2023-2/obligatorio-grupo11-bandera-caffarena-gonzalez

Sebastián Bandera-220417

Martín Caffarena-230351

Nicolas Gonzalez-231843

Entregado como requisito de la materia Ingeniería de software ágil 2

Índice

Designación de roles

De igual forma que en la etapa anterior comenzamos designando los distintos roles dentro del equipo, rotando así las responsabilidades de cada uno y permitiéndole a todos los miembros del equipo aprender desde distintas perspectivas. Para esta etapa se determinó que Martin sea el (SM), y Sebastian el (PO). Recordar que a su vez todos los miembros del equipo son Desarrolladores y Testers.

Informe de avance de la etapa

Para esta etapa se comenzó primero que nada con la designación de roles, de esta manera ya establecer una estructura clara dentro del equipo desde un inicio. Luego se comenzó de forma inmediata con el desarrollo de un Pipeline en GitHub, el cual primero el equipo debió aprender a desarrollar el mismo. Luego se determinó que issues serían los corregidos en esta etapa, sin embargo como se explica en el apartado “Elección de Bugs”, estos fueron cambiando a medida que avanzamos en la etapa. Por lo que también se debió de hacer una nueva instancia de Test Exploratorios por parte de todo el equipo, esto con el fin de poder encontrar nuevos Bug´s relevantes, apuntando más a Bug´s en la parte del Backend. A medida que estos fueron encontrados se fueron documentando de manera correspondiente y se agregaron al tablero. En total se resolvieron 3 Bug´s los cuales todos fueron resultados utilizando TDD (Especificado en apartados “Elección de Bugs” y “Reparación de Bug´s con TDD”).

Registro de las actividades realizadas (fecha, horas, integrante)..

En la etapa anterior en la reunión de retrospectiva el equipo concluyó que el uso de la bitácora había sido muy positivo y había aportado al seguimiento de horas, por lo que nuevamente para esta etapa esto se mantuvo. Sin embargo, no consideramos relevante encontrar otra herramienta para llevarla a cabo.

***Se utilizó como período de tiempo mínimo, 30 minutos = 0,5 horas**

Fecha	Concepto	Horas Bandera	Horas Caffarena	Horas Gonzalez	Total esfuerzo	Total día
23/09/2023	Creación de pipelines backend y frontend	2,0	0,0	0,0	2,0	
23/09/2023	Documentación pipeline	0,5	0,0	0,0	0,5	2,5
25/09/2023	Documentación elección de Bugs	0,0	0,5	0,0	0,5	
25/09/2023	Reunión	1,5	2,5	2,5	6,5	
25/09/2023	Test exploratorios	0,5	0,5	0,5	1,5	
25/09/2023	BUG Nro 22	2,0	0,0	0,0	2,0	
25/09/2023	BUG Nro 18	0,0	1,0	0,0	1,0	11,5
27/09/2023	BUG Nro 24	0,0	0,0	1,0	1,0	
27/09/2023	Reunión para desarrollo final de la documentación	1,5	2,5	2,5	6,5	
27/09/2023	Grabacion Retro y revisión de Bugs	1,5	1,5	1,5	4,5	12
	Entrega 2	0,0	0,0	0,0	0,0	26

Total: 26 horas/persona de trabajo.

Dificultades encontradas y formas de solución.

Desarrollo a nivel de Backend (visual Studio 2019)

En esta etapa al ser tan corta en duración no nos encontramos con grandes dificultades, sin embargo a inicios de esta nos encontramos con problemas para el manejo de la parte de backend del proyecto, esto llevó a una investigación de la causante del problema y se concluyó que esto se debía a la versión utilizada para desarrollar del Visual studio, en este caso 2019. Por lo que para poder solucionarlo debimos pasarnos al uso de Visual Studio 2022, luego de esto no nos encontramos con mayores dificultades.

Lecciones aprendidas y mejoras en el proceso

En esta etapa el equipo no cree haber aprendido ninguna lección que valiese la pena detallar, consideramos que esto tal vez se dio por razones de duración de la misma.

Nuevos aprendizajes

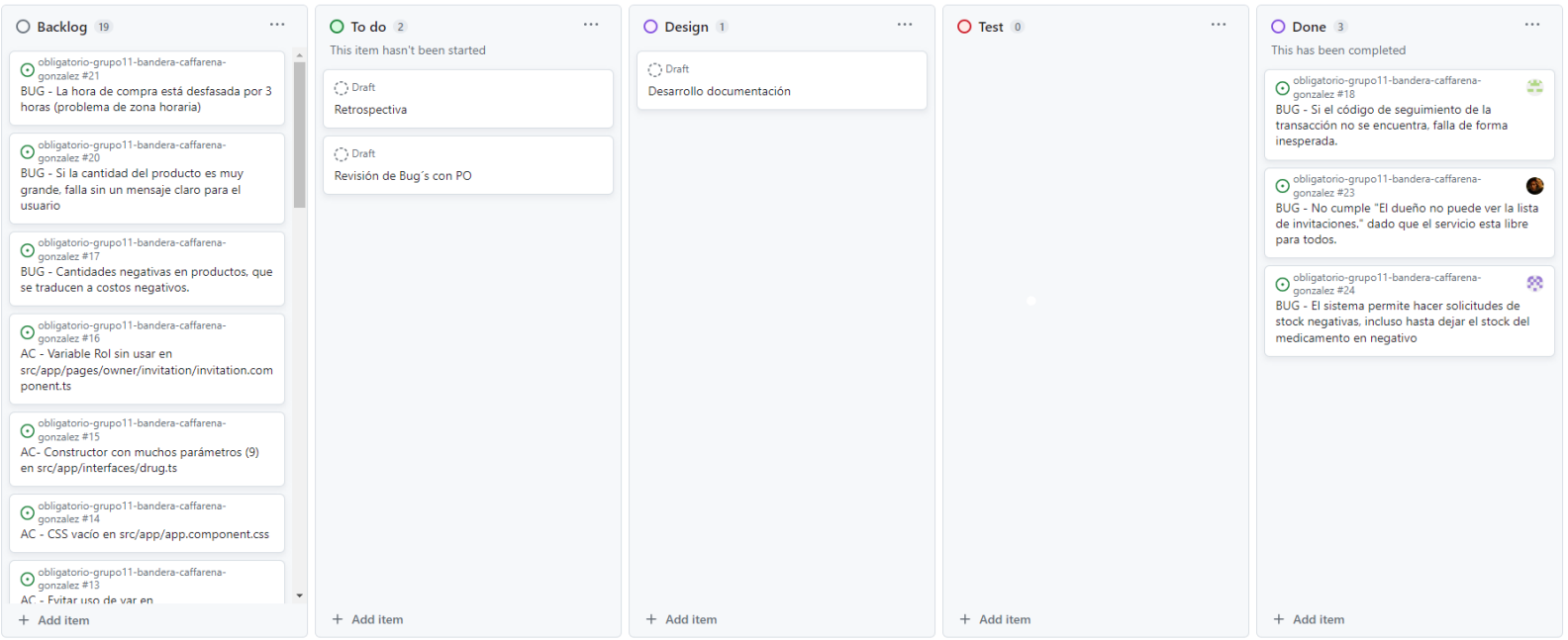
Algo que el equipo debió aprender fue la utilización del Pipeline en GitHub, ya que si bien todos los miembros del equipo estaban al tanto de esta tecnología y sabían lo que era un Pipeline, ninguno de los miembros había configurado uno por su cuenta.

Por lo que debimos aprender esta nueva técnica y con los conocimientos aprendidos en clase pudimos llegar a la configuración de Pipeline (Desarrollo en “Pipeline en GitHub”), el cual se ejecuta al pushear en la rama main y le permite al equipo un desarrollo más seguro el cual es verificado por dicho Pipeline, y nos permite el experimentar con los conocimientos aprendidos en clase.

Proceso de ingeniería

Como únicas actualizaciones al proceso de ingeniería tenemos que en esta etapa se definió la metodología para el manejo de ramas/commits y trabajo del equipo y se reestructuró el tablero de Kanban.
Para esto el equipo decidió utilizar el método de Trunk based, donde todos trabajamos en main y se comitea sobre el mismo, para luego ser corrido el Pipeline automático. Debido a que no se trabajó en el proyecto todos los días, debido a el tamaño de la entrega se decidió el hacer commit luego de cada día de trabajo individual, por lo que no se commitea a main todos los días sino solo aquellos en los que el desarrollador trabajo.

Tablero KANBAN



Para esta etapa se decidió hacer un cambio en el tablero de Kanban, viendo el contenido de la etapa se cambiaron las columnas para tener un flujo de trabajo más acorde al trabajo a realizar. Se quitó la columna “In Progress” dando lugar a dos nuevas columnas. La columna de “Design” en la cual las tarjetas correspondientes a issues iban a estar mientras se estaba

desarrollando el código para arreglar el mismo y también se encontraban aquellas tareas las cuales no corresponden a un issue pero estaban en progreso (como se puede ver en la imagen en este momento cuando se está desarrollando la documentación). También se agrega la columna de “Test”, en la cual estarían las tarjetas correspondientes a issues los cuales estarían ya implementados y deben ser testeados tanto por los testers como por lo el pipeline definido dentro del repositorio, las tareas no pasarían por esta columna e irían directamente a la columna “Done” luego de finalizadas.

Pipeline en GitHub

En GitHub se realizaron dos pipelines en la rama main para realizar el build en distintos ambientes, ubuntu, macOS y windows. Si una pipeline falla, se debe atender el error inmediatamente, ya sea revirtiendo el commit a un histórico correcto, o realizando la corrección. Además es ideal aprovechar esa oportunidad para aprender del error.

La pipeline para el backend, se ejecuta cuando algún archivo .cs o .csproj es modificado. Con .NET 6.0.x ejecuta la compilación y testeo en los tres sistemas antes mencionados. Fue necesario indicar la ubicación del proyecto en .NET con ‘Codigo/Backend/PharmaGo.sln’ dado que en caso contrario se buscaba en la raíz. Además se colocó ‘--property WarningLevel=0’ porque encontraban más de 350 advertencias que detenían la pipeline.

La pipeline para el frontend se ejecuta cuando hay algún cambio en ‘Codigo/Frontend/**’, con esto se practicó dos formas de restringir la pipeline (según subcarpeta o según extensión de archivo en el caso de la pipeline del backend). En el caso del frontend, no se encontraron pruebas unitarias, por lo cual la pipeline sólo realiza el build. Fue necesario realizar un ‘cd Codigo/Frontend’ antes de ejecutar los comandos para que el build se realice en el directorio correcto. La versión de node utilizada es la 16.x. Se removió el control del caché de dependencias ‘if: steps.cache-nodemodules.outputs.cache-hit != 'true'’ sobre ‘Install Dependencies’ porque provocaba error en las ejecuciones de la pipeline que no fueran la primera, lo cual era un gran problema de reproducibilidad.

Por lo tanto el pipeline se ejecutará siempre que se realice un push sobre la rama “main” la cual es nuestra rama de producción, con esto nos aseguramos que cada integración sobre nuestro ambiente de trabajo está controlado, es decir que se tiene feedback inmediato sobre la integración por si algún cambio genera un problema sobre alguna parte del sistema. Esto se ve en el tablero cuando las issues están en fase test estando pasando por el pipeline.

Elección de bugs

Para esta etapa se decidió el resolver 3 BUG´s, teniendo en cuenta la prioridad de estos asignada por el PO en la etapa anterior. En un principio se habían elegido 3 bugs los cuales tenían la prioridad más alta, sin embargo el equipo luego descubrió que estos eran bugs únicamente de la parte de Frontend y como se debía de resolver bugs de backend los cuales se desarrollaran con el uso de TDD, debimos de buscar nuevos bugs a resolver. Encontrando nuevos y eligiendo finalmente los siguientes bugs:

- **BUG - El sistema permite hacer solicitudes de stock negativas, incluso hasta dejar el stock del medicamento en negativo#24**

Severidad: Mayor
Prioridad: Inmediata
Asignado: Nico Gonzalez
Descripción: Se pueden realizar peticiones con cantidad negativa y el dueño aceptar las mismas, incluso de puede dejar los stock de los medicamentos en cantidad negativo

- **BUG - Si el código de seguimiento de la transacción no se encuentra, falla de forma inesperada.#18**

Severidad: Leve
Prioridad: Baja
Asignado: Martin Caffarena
Descripción: Si el código de seguimiento de la transacción no se encuentra, ocurre un null pointer en el backend y al usuario se le muestra lo siguiente

- **BUG - No cumple "El dueño no puede ver la lista de invitaciones." dado que el servicio está libre para todos.#23**

Severidad: Mayor
Prioridad: Media
Gravedad: No aplica
Asignado: Sebastian Bandera
Descripción: El endpoint https://localhost:44361/api/Invitations/{id} está libre, con lo cual no cumple "el dueño no puede ver la lista de invitaciones.

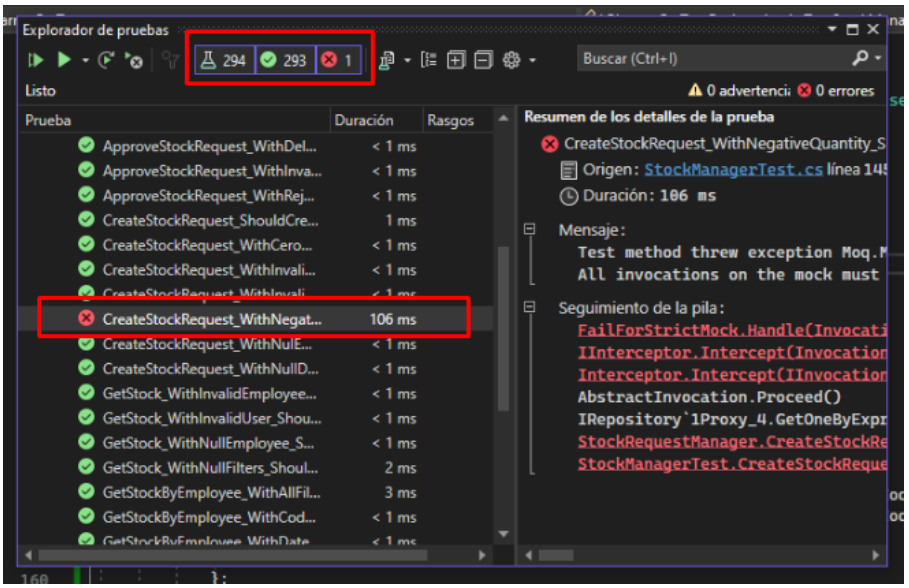
Reparación de bugs con TDD

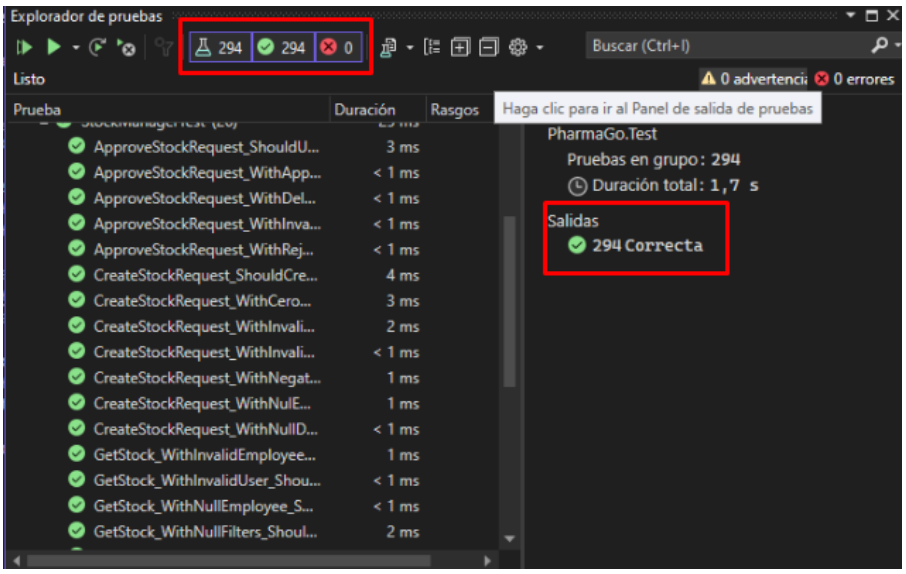
El equipo decidió utilizar la técnica de TDD para la reparación de bugs, la misma consiste en 3 etapas (RED, GREEN, REFACTOR). La primer etapa se crea el test para el código que se va a implementar pero sin implementarlo todavía por lo tanto el mismo falla (se ve como el test está en estado fallido en rojo), luego la siguiente etapa consiste en realizar la implementación mínima del código para que la prueba pase (se ve con estado verde la prueba ya que esta fue implementada como se esperaba) y por último se trata de mejorar el código sea con mejoras funcionales o sintácticas.
Para identificar estas etapas se decidió que luego de la acción en el commit (ya explicado en la anterior entrega) se indica en qué fase se encuentra este y por último sobre qué funcionalidad. Ejemplo: **UPDATE - RED - CreateStockRequest.**

El desarrollo bajo esta metodología de trabajo le permite al equipo avanzar de forma consistente y constante, ya que TDD si realizado de forma correcta permite el siempre estar “Parados en suelo firme” es decir, como para todo el código que desarrollamos ya hay test, es más difícil cometer errores y que estos sean pasados por altos, pudiendo así llegar a producción. Por lo tanto el equipo puede seguir desarrollando sobre esto sabiendo que las pruebas pasan y que no hay errores o al menos no errores catastróficos ocultos que podrían “explotar” luego en producción.

Evidencia de ejecución de casos de prueba

Aquí adjuntamos imágenes de los test corriendo , primero que nada vemos un caso en el que se estaba realizando TDD por lo que el Test estaba en Fase RED y luego podemos ver como luego de ya pasada esta fase nos encontramos en la fase GREEN en la que ya corren todos los Test, en algunos casos también había una instancia de REFACTOR, pero esta no se ver reflejada en los test sino más bien en las mejoras del código, tanto del código desarrollado como en el código de las pruebas.





Video de la revisión de los bugs con el PO

El vídeo de la revisión con el PO, se puede encontrar en el repositorio en GitHub, rama main, directorio “Documentación/Entrega 2 - ISA2”. Contiene la revisión de los tres bugs concatenados.

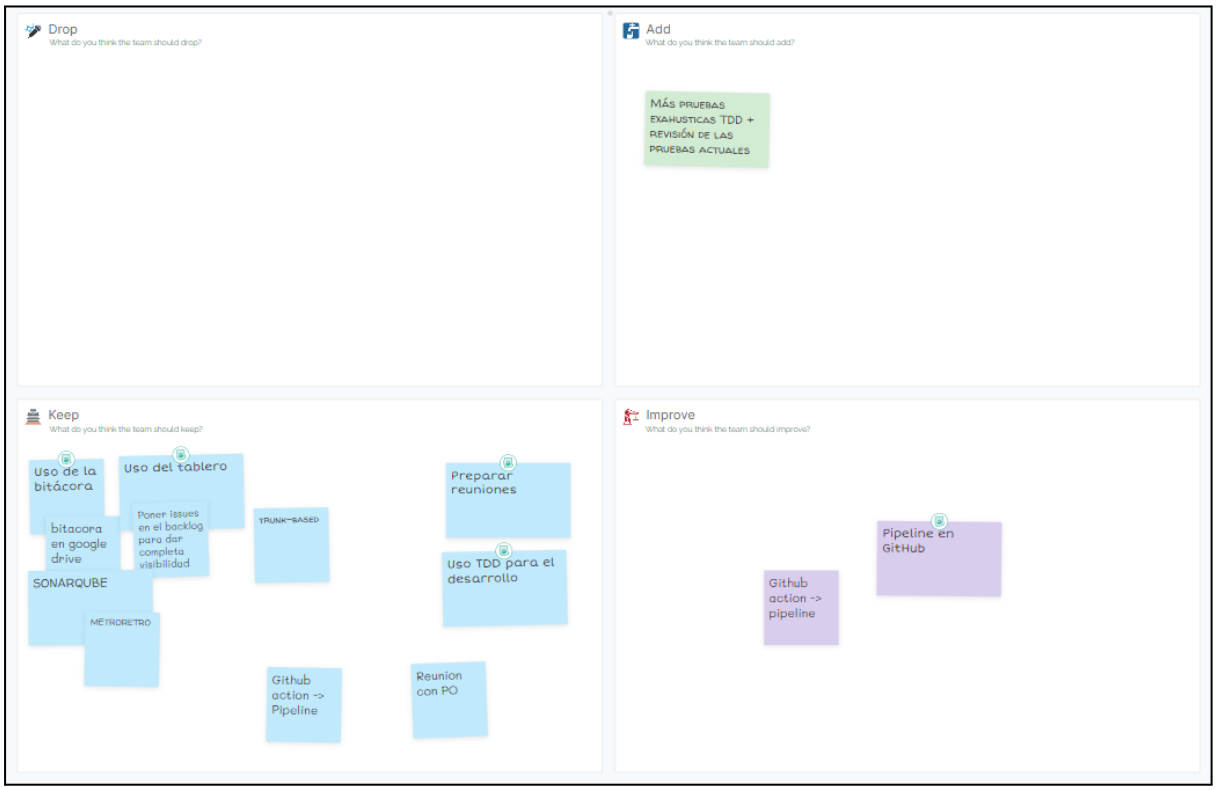
El link directo es:
https://github.com/IngSoft-ISA2-2023-2/obligatorio-grupo11-bandera-caffarena-gonzalez/blob/main/Documentaci%C3%B3n/Entrega%202%20-%20ISA2/RevisionesConPO.mkv

Video de retrospectiva

El vídeo de la retrospectiva se puede encontrar en el repositorio en GitHub, rama main, directorio “Documentación/Entrega 2 - ISA2”.

El link directo es:
https://github.com/IngSoft-ISA2-2023-2/obligatorio-grupo11-bandera-caffarena-gonzalez/blob/main/Documentaci%C3%B3n/Entrega%202%20-%20ISA2/Retro2.mkv

Tablero DAKI



DAKI - Imagen 1/2

DAKI - Imagen 2/2

En conclusión de la retrospectiva podemos entender que nuevamente se realizó un buen trabajo en esta etapa y aún mejor que en la etapa anterior, más organizado y mejor

estructurado, esto nuevamente se vio reflejado en la retrospectiva viendo que no se colocaron aspectos en “DROP” y se colocaron menos en “ADD” e “IMPROVE” con respecto a la etapa anterior. Se confirmó además que aquellas acciones a tomar a las que se llegaron en la primer retrospectiva, fueron cubiertas en su mayor parte, y se agregaron algunos nuevos temas a tener en cuenta, como el mantener y mejorar el Pipeline y/o realizar una investigación más a fondo de las pruebas de TDD ya existentes en el proyecto.