Universidad ORT Uruguay

Obligatorio - Informe Académico

Ingeniería de Software Ágil 2

Docentes:

Martín Solari

Carina Fontán

Sofía Decuadra 233397

Agustín Ferrari 240503

Joaquín Meerhoff 247096

https://github.com/ORT-ISA2-2022S1/obligatorio-decuadra_ferrari_meerhoff.git

2022

Indice

Introducción

El presente informe tiene como objetivo funcionar como resumen general del proyecto, teniendo en cuenta los ítems mencionados en la definition of done presente en aulas. Se detallan las actividades realizadas y se reflexiona sobre las decisiones tomadas para cumplir los objetivos del proyecto, manteniendo un enfoque en la perspectiva de DevOps.

1. Resumen de gestión del proyecto

Al principio del proyecto se tomó la decisión de seguir un marco de gestión ágil con elementos de Scrum como los artefactos y ciertas ceremonias, siendo estas el sprint planning donde se decidió el sprint backlog, la sprint review (que fueron grabadas[1]) en la cual evaluamos como product owners el incremento del producto realizado por los desarrolladores y al final la sprint retrospective donde se reflexionó siguiendo el formato DAKI (Drop, Add, Keep, Improve) sobre cómo se trabajó en el sprint, y como mejorar en siguientes iteraciones.

Se desarrolló siguiendo Trunk Based Development (TBD) en github, por lo cual se buscó hacer uso de ramas cortas integradas a main en menos de un día desde que se crearon. Para favorecer la posibilidad de desarrollar por separado se tomó la decisión de dejar a discreción del dueño de la rama el hacer una review en los pull requests, evitando cuellos de botella en caso de no estar disponible el reviewer.

En el primer sprint se analizó el código presente y documentó cómo se seguiría el proyecto en las siguientes iteraciones. En el segundo sprint se enfocó en el desarrollo de nuevas funcionalidades siguiendo BDD haciendo uso de nuevas herramientas y en el tercero se terminaron de implementar las funcionalidades del sprint anterior con herramientas de testing funcional automatizado. Es esperable entonces que el esfuerzo de los sprints fue aumentando como se puede notar en la tabla 1, ya que cada sprint incluyó actividades de sprints anteriores además de nuevas tareas que requirieron aprender nuevas herramientas.
Un aspecto notable en la tabla 1 es la gran diferencia entre el esfuerzo estimado y real, consideramos que este se vió afectado por actividades agregadas a mitad de sprint por el feedback recibido, además de no tomar en cuenta el esfuerzo necesario para aprender a usar las herramientas.

[1] Links a videos: Sprint 2: https://www.youtube.com/watch?v=xDyq6rzCfdE
Sprint 3: https://www.youtube.com/watch?v=e0VnvqfJmUA

Tabla 1: Horas persona en sprint por sección

Sprint	Rango de días	Planificación	Ejecución	Control	Estimado	Total
1	25/04 - 08/05	6	49.5	12	36	67.5
2	09/05 - 22/05	14.4	40.9	16.5	50	71.8
3	25/05 - 02/06	12	57.55	26.5	40.5	96.05
Total	**25/04 - 02/06**	**32.4**	**147.95**	**55**	**126.5**	**235.35**

Actividades y métricas

La siguiente tabla presenta las actividades realizadas con hipervínculo a su entregable en el repositorio si corresponde. Con respecto a las funcionalidades agregadas, siendo estas el alta, baja y obtención de puntos de carga, su entregable es la funcionalidad en sí.

Tabla 2: Horas persona por actividad

Título	Esfuerzo Sprint 1	Esfuerzo Sprint 2	Esfuerzo Sprint 3	Esfuerzo Total
Guía del proceso de ingeniería y mantenimiento del proyecto	3.5	-	1.5	5
Informe deuda técnica	3.5	-	1.5	5
Guía del desarrollo con BDD	-	1	1	2
Guía de la pipeline del proyecto	1	-	1.5	2.5
Análisis de métricas	0.5	0.5	4	5
Alta de puntos de carga	-	12.5	19.5	32
Baja de puntos de carga	-	6.5	11	17.5
Obtener puntos de carga	-	-	7.3	7.3
Issue: Reserva en mismo día	-	2	-	2
Issue: Cambio de claves de administradores	-	1	-	1
Issue: Datos innecesarios en base de datos	-	2	-	2
Total	**8.5**	**25.5**	**47.3**	**81.3**

Lead Time y Cycle Time to Review

Como se explica en el reporte del sprint 3, por no mover a Done las tareas hasta la sprint review al final del sprint, consideramos más informativa la métrica de Lead Time y Cycle Time to review que Lead Time y Cycle Time, ya que nos informan del tiempo necesario de desarrollo hasta que esperan ser revisadas.

Sprint 2 **Sprint 3**

Tabla 3: Lead Time to Review

Tarea	Días en desarrollo
Agregar punto de carga	11
Eliminar punto de carga	10

Tarea	Días en desarrollo
Agregar punto de carga	7
Eliminar punto de carga	9
Obtener puntos de carga	4

Tabla 4: Cycle Time to Review

Tarea	Días en desarrollo
Agregar punto de carga	5
Eliminar punto de carga	4

Tarea	Días en desarrollo
Agregar punto de carga	3
Eliminar punto de carga	5
Obtener puntos de carga	3

Issues

Desde un comienzo se detectó y reportó issues en la plataforma de github, discriminando por tipo y prioridad en tres niveles: bajo, medio y alto. La distribución por estas categorías se puede ver en la tabla 3, donde se puede notar que se han reportado más issues de los que se han cerrado, principalmente de prioridad media y baja, del tipo “bug” asociado al código.

A partir del segundo sprint se comenzó a resolver issues, tomando los de mayor prioridad del sprint anterior. Esto llevó a las actividades de resolver un issue asociado a no poder reservar un hospedaje en el día que se esté reservando y otro asociado a datos que quedaban en la base de datos a pesar de ser borrados. Se descubrió que el segundo era un problema causado por los datos de prueba por lo cual se sustituyó con un bug asociado a no poder cambiar la clave administradores.

Continuamos estas prácticas de reportar todo issue encontrado y resolver aquellos de mayor prioridad a lo largo del sprint 2 y en el sprint 3, independientemente de si surgieran por nuestras adiciones al código o estuvieran presente en el material del obligatorio.

Tabla 5: Resumen de issues por sprint

Cantidad por label	Sprint 1	Sprint 2	Sprint 3	Total
Creados en Sprint	20	3	11	34
Abiertos en Sprint	13	1	8	22
Cerrados en Sprint	7	2	3	12
Tipo - Documentación	6	0	0	6
Tipo - Configuración	1	1	2	4
Tipo - Bug	15	0	4	19
Tipo - Duplicados	0	1	0	1
Tipo - Inválidos	0	1	7	8
Prioridad Baja	13	0	5	18
Prioridad Media	6	1	6	20
Prioridad Alta	1	1	0	2

2. Reflexiones sobre el aprendizaje

2.1 Aplicar un marco de gestión ágil.

Con respecto al marco de gestión ágil, notamos que algo que nos faltó fue herramientas de seguimiento de esfuerzo automatizado, ya que utilizamos github projects y este por si mismo no ofrece estas funcionalidades. Esto nos hizo perder mucho tiempo en calcular el esfuerzo manualmente. Esto sería un posible punto de mejora haciendo uso de herramientas más robustas como por ejemplo Toggl que no llegamos a poder utilizar en el proyecto.

Consideramos adecuado mantener el uso de ceremonias de scrum y las reuniones periódicas para mantener a todos los miembros actualizados sobre el progreso del proyecto.

Otro aspecto importante a mantener serían los elementos de TBD como las ramas cortas, ya que notamos que es más sencillo incorporar elementos si nos limitamos a partes pequeñas de a poco.

2.2 Analizar la deuda técnica.

Ya desde la primera iteración notamos la importancia de analizar la deuda técnica, ya sea con herramientas como NDepend de análisis estático de código, manualmente haciendo pruebas exploratorias o leyendo el código buscando elementos que no sigan clean code o que podrían ser mejorados con un

refactor. Por medio de estas técnicas se llegó a descubrir la mayor parte de los issues reportados, y es por esto que las mantendríamos en el proyecto, excepto por NDepend que nos vimos limitados por su versión de prueba de 14 días.

2.3 Implementar un repositorio y procedimientos de versionado.

A este punto le dimos una gran importancia desde el comienzo, en particular en mantener organizado (con READMEs y documentos que aclaren el contenido) y estandarizado (ya sea en los commits, pull requests e issues con herramientas de templates) el repositorio facilitando la visibilidad no solo para los miembros del equipo, sino también para figuras externas.

Mantendríamos el uso de releases para marcar estados finales de sprints para tener un histórico si fuese necesario en un futuro, y el requerimiento de hacer un pull request antes de mergear a la rama main con el mismo propósito de dejar visible la evolución del proyecto.

De todos modos, consideramos que algunas decisiones tomadas al principio afectaron el uso del repositorio, como archivos en lugares externos (HackMD.io) que requirieron ser sincronizados manualmente. Un aspecto a mejorar sería definir un estándar más completo de codificación que el actual, incluyendo aspectos como máxima lineas por métodos. Se tomó la decisión de no realizarlo porque implicaría modificar todo el código presente para seguir el nuevo estándar, incluyendo el material proporcionado.

2.4 Crear un pipeline con eventos y acciones.

Un elemento importante del desarrollo que no tuvimos tan presente a lo largo de la carrera fue la pipeline de desarrollo con eventos y acciones. Notamos que la presencia de esta tuvo un gran impacto en la confianza de incorporar nuevas funcionalidades y desarrollar en ramas separadas al tener un feedback rápido, haciendo uso de los tests unitarios. Otra parte importante de la pipeline es la checklist previo a un merge para recordar hacer uso de los linters, correr las pruebas de funcionalidad, además de seguir los estándares de programación definidos.

Algo que se le podría agregar sería el bloquear todo merge que no pase los controles definidos, además de un posible servidor al que desplegaría la pipeline, probablemente en la nube como Amazon AWS, incluyendo entonces el aspecto de despliegue de CI/CD.

2.5 Configuración de entornos tipo producción.

A pesar de haber tenido problemas con los entornos de desarrollo al principio del proyecto, no se definió hasta más adelante las versiones específicas de las herramientas y frameworks necesarios. Consideramos que esto podría ser un aspecto a mejorar, llevando a que nuevas herramientas requieran una especificación de versiones a utilizar.

2.6 Integrar prácticas de QA en el pipeline y gestionar el feedback.

Opinamos que el feedback es un aspecto importante para estas entregas y que debe tomarse en cuenta cuando se recibe sin importar el momento, incluso si se está a mitad de un sprint. Como feedback entendemos no solo las devoluciones a previas entregas, sino también aspectos como métricas de resultados de pruebas automáticas, de issues abiertos y cerrados por sprint, de Lead Time y Cycle Time to Review (ya que no se mueven a Done las tareas hasta que pasen por el sprint review).

Algo que podría haberse incluido es un template y directorio en el repositorio donde se documenta la recepción del feedback y el análisis de este, para no solo incluir esta información en los informes de avance.

Algunas prácticas que mantendríamos de QA sería el uso de Linters (que podrían ser configurados mejor de lo que lo hicimos), realizar testing exploratorio, analizar el código rutinariamente, hacer checklists previo al merge y la mejora continua del pipeline.

Todas estas prácticas nos permiten tener un proyecto más robusto, ya que ayudan a encontrar errores y evitar que nuevos cambios rompan elementos que estaban funcionando, como en el caso de los tests automáticos que pueden funcionar como tests de regresión. Además reducimos la cantidad de deuda técnica que generaría el proyecto sin estas prácticas.

2.7 Automatizar el testing.

Otro importante aspecto del proyecto que no habíamos realizado hasta este obligatorio fue el desarrollo de código siguiendo Behaviour Driven Development (BDD) al escribir historias de usuario y sus escenarios para luego utilizar estos como testing funcional automático. Consideramos que fue una buena adición a las herramientas de testing y lo mantendríamos en un proyecto de este estilo pero que deberíamos mejorar como los escribimos, ya que al implementarlos para el frontend, notamos que estos habían sido escritos con el backend en mente y no desde el punto de vista del negocio, por lo cual debieron ser re-escritos.

Las herramientas que utilizamos para realizar estas pruebas fueron Specflow y Selenium y si bien tuvimos problemas con ambas (en Specflow el escribir los escenarios y en Selenium el independizar las pruebas), creemos que con la experiencia adquirida su uso podría ser beneficioso para futuros proyectos y se podría evitar problemas que nos encontramos en esta instancia.

Si bien ya teníamos experiencia con el testing funcional, notamos que la automatización del mismo fue beneficiosa para el proyecto, evitando regresiones en funcionalidad cuyas pruebas hubieran sido manuales, tediosas, y propensas a errores humanos.

Por ser la primera vez que hicimos uso de estas, consideramos que algunos aspectos a agregar serían la posibilidad de correrlos en github actions con un workflow recurrente, no en cada commit sino diario o menos frecuente.

3. Lecciones aprendidas

- ***Ser consciente de herramientas que funcionan de manera no uniforme en todas las estaciones de trabajo:*** Ej: Se hizo uso de herramientas que funcionan diferente dependiendo del entorno.
- ***Es mejor estimar el esfuerzo para actividades de más que de menos:*** Se pueden pasar por alto detalles de tareas y subestimar algunas por el desconocimiento de herramientas nuevas, lo que puede perjudicar la administración.
- ***Definir estándares temprano y actualizarlos:*** Un proyecto ordenado que sigue estándares facilita la lectura.
- ***La importancia de las métricas y las herramientas que nos permiten analizarlas:*** Calcular las métricas manualmente desfavorece el uso de estas, y las hace propensas a errores.
- ***La importancia de trabajar simultáneamente, incluso si es en tareas diferentes, para poder hacer uso del Andon Cord:*** Favorece el tener el proyecto siempre en un estado funcional y entregable.
- ***Desarrollar en ramas cortas que sean integradas en menos de un día:*** Evita integraciones demasiado grandes para analizar, y también reduce la posibilidad de un merge conflict difícil de arreglar.
- ***La importancia de un histórico de la evolución del proyecto:*** En caso de errores se puede tener un claro log de donde fueron agregados, ya sea en los releases, pull requests o artefactos de workflows en github.
- ***Una única fuente de verdad (Single Source of Truth):*** Tener documentación, código y configuraciones juntas en un repositorio central facilita el análisis de viejas versiones.
- ***La necesidad de una pipeline bien definida para el mantenimiento:*** El uso de github actions, checklists para partes manuales, etc. promueven las prácticas de CI/CD en el equipo y favorecen los controles detectivos.

4. Conclusiones

El adoptar un enfoque DevOps lleva a la mejora de calidad del producto, haciendo uso de prácticas de mantenibilidad para mantener baja la deuda técnica y permitir el desarrollo continuo con la seguridad, dada por las pruebas y su feedback, de que siempre se esté en un estado entregable.
En conjunto con Trunk Based Development se reduce el tamaño de las integraciones permitiendo encontrar problemas rápidamente, y en el caso de que surja un problema mayor, se pueda hacer uso adecuado del Andon Cord y resolverlo en equipo.

El uso y automatización de pruebas funcionales escritas desde la perspectiva del cliente o usuario disminuye la posibilidad de que ocurran malentendidos en lo esperado de funcionalidades existentes y nuevas. Además, estas pruebas funcionales pueden ser usadas para controlar regresiones, volviendo más robusta la solución.

Todo esto nos conduce a un proyecto mantenible, que favorece el mantener baja la deuda técnica, con enfoque en facilitar la colaboración entre los miembros del equipo, y en un contexto corporativo, este conjunto de resultados proporcionaría una ventaja competitiva.
Para un futuro, los aprendizajes y conocimientos adquiridos para esta entrega van a tener un impacto en la forma en que gestionaremos, desarrollaremos y mantendremos en próximos proyectos tanto en el resto de la carrera como en nuestra vida laboral.

5. Guía de instalación y despliegue en producción

Asumiendo que se tienen las versiones indicadas de los frameworks en el documento EnvironmentGuide.md del repositorio, los comandos descritos en la sección Getting Started del README.md del repositorio permiten correr el Frontend, Backend y pruebas de integración del proyecto.
Para el despliegue de la aplicación, como no se han realizado cambios que afecten el proceso de build del frontend y backend, la guía de despliegue presente en el material del obligatorio no se ha visto afectada.

	RUN	BUILD	LINT

Backend	`cd ./Source/MinTurBackend`
	`dotnet restore`
	`dotnet build`
	`cd ./MinTur.WebAPI`
	`dotnet run`
		`cd ./Source/MinTurBackend`
		`dotnet restore`
		`dotnet build`
			`cd ./Source/MinTurBackend`
			`dotnet build`

Frontend	`cd ./Source/MinTurFrontend`
	`npm install --force`
	`npm run start`
		`cd ./Source/MinTurFrontend`
		`npm install --force`
		`npm run build:prod`
			`cd ./Source/MinTurFrontend`
			`npm install --force`
			`npm run lint`