Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio - Ingeniería de Software Ágil 2

Link al repositorio

Entrega 1

Integrantes:
Danilo Biladoniga - 231749
Tomás Núñez - 257564
Germán Oller - 242312

Docentes:
Carina Fontan
Alvaro Ortas

Índice

Definición del proceso de ingeniería en el contexto de KANBAN

Origen

Kanban, en japonés “letrero” o “cartel de publicidad”, es un proceso de fabricación de productos que nos permite realizar entregas constantes y continuas de valor a los clientes. Se utiliza en el entorno de definición y mejora de procesos desde la década de 1950.

Fue desarrollado y aplicado por primera vez por la empresa Toyota como sistema de programación para la fabricación JIT (Just In Time).

Representa un sistema con un comportamiento “pull”, puesto que el proceso de fabricación depende del proceso de consumo mediante pedidos. En otras palabras, la producción se realiza en base a la demanda de los clientes.

Kanban como metodología ágil

En el proceso de ingeniería se planifica, diseña, desarrolla, prueba y se entrega un producto o servicio, cuyo principal objetivo es optimizar la eficiencia, calidad y capacidad de respuesta de los equipos de trabajo.

Comprende el Ciclo de Deming, comúnmente conocido como su acrónimo “PDCA” o también “Plan”, “Do”, “Check”, “Act”.

Así mismo, determina 6 principios para su implementación exitosa:

- Visualizar el flujo de trabajo
- Limitar el WIP
- Gestionar y medir el flujo
- Implementar ciclos de feedback
- Explicitar políticas y procedimientos
- Mejora continua mediante la colaboración

Roles dentro del equipo

Hemos establecido tres roles en el equipo:

- Product Owner:

Responsable de definir y priorizar los elementos del backlog de un producto de software. Así mismo debe evaluar el feedback de los clientes y en base a eso priorizar las tareas para el equipo de desarrollo, de esta forma poder cumplir con las necesidades más valiosas de los clientes.

- Desarrollador:

Responsable de implementar código y desarrollar las características y funcionalidades definidas así mismo como encargado de seguir buenos estándares de codificación y mantenimiento del código.

- Tester:

Responsable de la planificación y ejecución de pruebas, así como también documentar los informes correspondientes a los errores encontrados y validación de entregables.

Estos roles fueron distribuidos de la siguiente manera:

Miembro	Rol/es
Danilo Biladóniga	Developer, Tester
Tomás Núñez	Developer, Tester, Product Owner
Germán Oller	Developer, Tester

Ceremonias

Consisten en una serie de reuniones entre los integrantes del equipo para asegurarse que las tareas y los procesos se desarrollen de manera óptima. Entre las distintas ceremonias se incluyen las stand ups, que permiten al equipo visualizar cuellos de botella así como también saber en que está trabajando cada miembro. Por otro lado, las retrospectivas nos permiten identificar impedimentos en el flujo de trabajo, así como posibles mejoras.

En nuestro caso hemos realizado una kick off, la cúal nos sirvió para establecer cómo íbamos a trabajar a lo largo de estás primeras dos semanas. Además realizamos dos stand ups para compartir las actividades del equipo, así como ayudarnos en caso de que exista algún bloqueo que no nos permita continuar con nuestras tareas. Por último hicimos una retrospectiva siguiendo el método DAKI.

Explicación del tablero y su vínculo con el proceso de ingeniería

Diseñar el tablero Kanban

Como lo indica el primer principio, antes de empezar a trabajar debemos diseñar un tablero Kanban que nos permita visualizar el flujo de trabajo. Existen 2 tipos de tableros: “Agile” y “Sustentable”.

En el caso del tablero “Agile” se definen columnas como etapas del proceso, las cuales incluyen “To Do”, “In Progress”, “Testing” y “Done”.

Por otro lado en un tablero “Sustentable” se suelen agregar más etapas al proceso como “Planificación”, “Entrega” o “Review”.

Nuestro tablero

Como el objetivo de esta primera entrega es encontrar y reportar issues, decidimos utilizar un tablero de Kanban simple con 3 columnas.

To do	Doing	Done

En la columna “To do” fuimos agregando las distintas tareas que tuvimos que realizar relacionadas al proyecto.

La columna Doing la utilizamos para indicar a los miembros del equipo que tareas se estaban realizando.

Finalmente, cuando una tarea era finalizada se movía a la columna Done.

Las tareas tienen una prioridad (alta, media o baja), que representa la importancia que creíamos que tenían al momento de agregarlas al tablero. También tienen 2 campos de fechas, para registrar cuando se comenzó a trabajar en la tarea y la fecha en la que se finalizó. Todas las tareas estaban asignadas a al menos 1 persona, pero podían estar asignadas a más.

Creación y mantenimiento del repositorio

Repositorio

El proyecto está respaldado en un repositorio en GitHub, lo cual nos permite trabajar de forma colectiva y realizar un control de versiones.

Tenemos 2 carpetas principales: Documentación y Código.

La primera está dividida en EntregasISA2, donde guardamos los reportes de las diferentes entregas, y en AplicacionDA2, que contiene lo referido a la documentación de la aplicación.

En la segunda carpeta tenemos todo el código de la aplicación, dividido en Frontend y Backend.

Manejo de versiones

Para el manejo de versionado utilizaremos el modelo de integración Trunk-based, ya que favorece la integración continua. Consiste en tener una única rama principal, desde la cual se crean feature branches pequeñas y de corta duración.
También decidimos utilizar Pull requests para los merges de código con su correspondiente revisión previa.

Branches

Desde la rama principal crearemos ramas que deberán cumplir con el siguiente estándar de nomenclatura:

- En caso de estar solucionando algún bug o error presente en el código, se utilizará: fix-NombreDeLaRama

- Si se están agregando nuevas funcionalidades, la nueva rama se nombrará: feature-NombreDeLaRama

- Si solamente se quiere agregar documentación nueva, o modificar la existente, el nombre de la rama será: documentation-NombreDelArchivo. En caso de agregar varios archivos a una misma carpeta se puede indicar el nombre de la carpeta a la que pertenecen.

Cabe destacar que en todos los casos se utiliza UpperCamelCase.

Issues

Para reportar los bugs y errores encontrados en las distintas funcionalidades del proyecto usaremos la sección de Issues que proporciona GitHub.

Establecimos un formato que deben cumplir las issues reportadas:

- ID: [número de issue]: nos permite identificar cada issue de manera rápida.

- Descripción: se utiliza para dar información detallada acerca del issue.

- Pasos (aplica solamente en bugs): pasos a seguir para reproducir el bug.

- Tipo: la calificación fue la siguiente:
    - Bug: comportamientos que el software debería hacer bien y no lo hace o se omiten.
    - Análisis de código: defectos encontrados al momento de analizar el código.

- Severidad (solo se aplica en issues de tipo bug):
    - Crítico: un defecto que obstaculice o bloquee completamente la prueba o el uso de un producto o función.
    - Mayor: una función principal que no cumpla con los requisitos y se comporte de manera diferente a lo esperado, funciona muy lejos de las expectativas o no está haciendo lo que debería estar haciendo.
    - Menor: función que no cumpla con sus requisitos y se comporte de manera diferente a lo esperado, pero su impacto es insignificante hasta cierto punto o no tiene un impacto importante en la aplicación.
    - Leve: defecto cosmético.

- Prioridad (solo se aplica en issues de tipo bug):
    - Inmediata: plazo máximo 24 hs.
    - Alta: plazo máximo 48 hs.
    - Media: plazo máximo un par de semanas.
    - Baja: sin plazo.

- Gravedad (solo se aplica en issues de tipo análisis de código):
    - Alta: errores de diseño que pueden conducir a problemas significativos de rendimiento o mantenibilidad.
    - Media: dificulta la comprensión del código o de un problema.
    - Baja: convenciones de codificación que no se siguen consistentemente.

- Esfuerzo: esfuerzo invertido en la tarea, medido en horas persona.

- Lead Time: tiempo desde que el issue se encuentra en TO DO hasta que llega a DONE.

- Cycle Time: tiempo que el issue se encuentra en DOING

Tags

Por otro lado, respecto a las tags hemos usado las siguientes:

- bug: representa un issue de tipo bug
- code: representa un issue de tipo análisis de código

Tags para niveles de gravedad

- gravedad alta
- gravedad baja
- gravedad media

Tags para niveles de prioridad

- prioridad alta
- prioridad baja
- prioridad inmediata
- prioridad media

Tags para niveles de severidad

- severidad crítica
- severidad leve
- severidad mayor
- severidad menor

Análisis de deuda técnica

Calidad de código

Basándonos en las métricas reportadas por los autores en la documentación del proyecto, podemos observar que no se cumple el principio de dependencias estables (SDP). El paquete IBusinessLogic tiene una inestabilidad de 0.56 y depende de ExportationModel, que cuenta con una inestabilidad de 0.61. Si bien la diferencia entre las inestabilidades no es demasiado grande, se debería buscar la forma para cumplir con el principio.

Otro detalle no menor es que la documentación no cuenta con un análisis de cohesión relacional, ni del cumplimiento de los principios de cohesión a nivel de paquetes.

Los autores manifiestan haber tenido problemas con la herramienta NDepend, utilizada para calcular las métricas, por lo que decidimos utilizar el analizador nativo de Visual Studio para verificar los resultados. Concluimos que, además del error previamente mencionado, no hay ninguna otra métrica alarmante.

Cobertura de pruebas unitarias

El código de frontend no cuenta con tests unitarios, solo fue probado funcionalmente. Esto puede suponer un problema en caso que algo relacionado no funcione como es debido.

Los autores afirman que, teniendo en cuenta “los archivos que contienen lógica”, llegan a un 91,28% de cobertura de backend. Sin embargo, analizando todo lo que debería tener tests asociados (esto es, archivos no autogenerados), vemos que la cobertura de código realmente es de 90,08%. Este resultado supera ligeramente el mínimo recomendado de 90%.

Análisis estático

Pudimos observar algunos problemas referidos al cumplimiento de estándar de nombres, al manejo de excepciones, tamaño de métodos, entre otros. Cómo ya mencionamos, decidimos reportar estos problemas utilizando el tag de code.

Registro de esfuerzo

Para llevar a cabo el registro de esfuerzo utilizamos una hoja de cálculo en Google Drive. En ella cada integrante del equipo tenía una tabla en la que registraba la tarea realizada, la fecha en la que la hizo, la hora de comienzo y la hora de finalización, para finalmente calcular la duración total de la tarea.
El resultado obtenido fue el siguiente:

Tomás Núñez				
Trabajo realizado	**Fecha**	**Hora Inicio**	**Hora Final**	**Horas**
Kick off	12/9	20:30	21:00	0:30
Instalación de la aplicación	12/9	21:00	22:30	1:30
Investagación formas de análisis de código	13/9	16:00	17:00	1
Standup	14/9	18:00	18:30	0:30
Análisis cobertura de código	14/9	20:30	21:30	1
Análisis calidad de código	15/9	16:00	17:30	1:30
Verificar testing unitario	15/9	18:00	19:00	1
Métricas	16/9	16:00	17:30	1:30
Documentación	17/9	17:00	18:30	1:30
Standup	17/9	18:30	19:00	0:30
Documentación y revisión general	18/9	16:00	17:00	1
Retrospectiva	18/9	21:10	21:40	0:30
			Total	12

Danilo Biladoniga				
Trabajo realizado	**Fecha**	**Hora inicio**	**Hora Final**	**Horas**
Kick off	12/9	20:30	21:00	0:30
Instalación de la aplicación	12/9	21:00	22:30	1:30
Standup	14/9	18:00	18:30	0:30
Realizar testeo exploratorio	15/9	11:00	13:00	2
Realizar testeo exploratorio	16/9	10:00	12:00	2
Realizar testeo exploratorio	17/9	14:00	18:00	4
Documentación	17/9	18:00	18:30	0:30
Standup	17/9	18:30	19:00	0:30
Retrospectiva	18/9	21:10	21:40	0:30
			Total	12

Germán Oller				
Trabajo realizado	**Fecha**	**Hora inicio**	**Hora Final**	**Horas**
Kick off	12/9	20:30	21:00	0:30
Instalación de la aplicación	12/9	21:00	22:30	1:30
Investigar Como Hacer Analisis de Codigo	13/9	17:30	18:30	1:00
Standup	14/9	18:00	18:30	0:30
Instalar NDepend	14/9	17:30	18:30	1:00
Analizar Codigo	16/9	19:00	21:00	2:00
Analizar Codigo	17/9	19:00	20:00	2:00
Standup	17/9	18:30	19:00	0:30
Documentacion	18/9	16:00	17:00	1:00
Retrospectiva	18/9	21:10	21:40	0:30
			Total	10:30