DISEÑO Y MANTENIMIENTO

DEL SOFTWARE

Entrega 2 – Proyecto:

Aplicación de tablones Kanban para Scrum.

Autores:

Yeray Sardón Ibáñez

Héber Saiz Campomar

Índice.

Contenido

Enunciado.

El proyecto se basa en una aplicación de tableros Kanban para Scrum, implementada en el lenguaje Java.

Requisitos de la aplicación en forma de historias de usuario:

1. Como evaluador quiero acceso al código y su historial para ver la implementación
2. Como evaluador quiero una memoria para evaluar el uso de buenas prácticas de diseño y patrones de diseño
3. Como evaluador quiero instrucciones para poder construir y lanzar la aplicación
4. Como cliente quiero poder añadir miembros al equipo
5. Como cliente quiero poder añadir sprint backlogs para las nuevas iteraciones
6. Como cliente quiero poder añadir tareas al product backlog
7. Como cliente quiero poder mover tareas del product backlog al sprint backlog
8. Como cliente quiero poder mover tareas entre las fases del ciclo de vida

9. Como cliente quiero poder editar las tareas
10. Como cliente quiero que los backlogs y tareas se mantengan cuando vuelva a abrir la aplicación
11. Como cliente quiero que la aplicación pueda tener nuevas formas de guardar el modelo en el futuro (implementar una si no se implementa el punto 12)
12. Como cliente quiero que la aplicación pueda tener nuevas formas de interacción en el futuro (implementar una si no se implementa el punto 11)

Estructura.

Nuestro proyecto consta de 3 paquetes acordes a la arquitectura Modelo Vista-Controlador y un paquete adicional para la persistencia.

El paquete Modelo contiene una multitud de clases que se corresponden con la estructura del programa. En el tenemos las siguientes clases e interfaces:

Grado – Enumeración que contiene los niveles que se pueden utilizar para indicar, por ejemplo, la gravedad de un defecto.

Estado – Enumeración en la que se encuentran los posibles estados de una tarea.

Backlog – Interfaz de la que heredan tanto ProductBacklog como SprintBacklog. En principio no tiene ninguna utilidad, pues no define ningún método, pero se decidió dejarla por si en el futuro se quisiese ampliar y definir nuevas funcionalidades.

ProductBacklog – Clase que extiende de Backlog e implementa las funcionalidades para crear el product backlog de la aplicación, donde se añadirán las tareas. Además, es el único sitio donde se pueden editar las tareas. Si una tarea se ha movido a un sprint, se entiende que cumplía las características para ello y, en caso de que no lo haga, se debería crear un defecto o historia de usuario nuevo que lo corrigiese.

SprintBacklog – Contiene La estructura para crear los sprints, con los métodos necesarios para acceder a los atributos. Una vez finalizado el sprint, no se podrá editar nada de él, dado que, al acabar la duración, para corregir o cambiar cosas se debería hacer en el sprint actual.

Requisito – Clase que contiene todos los datos de los registros, incluyendo su nombre o las tareas asociadas. Además, sirve como padre para las historias de Usuario y los requisitos.

Tarea – En Tarea se tiene toda la información referente a las tareas que se moverán en los backlogs y que conforman el eje de la aplicación. Las tareas se crean desde la clase Requisito, ya que cada una tiene que ir asociado a un requisito.

Defecto – Define todos los atributos y métodos que podría tener un defecto. Extiende de Requisito e implementa los campos propios de los defectos (Se han seleccionado unos pocos a modo de ejemplo).

HistoriaDeUsuario – Clase que extiende de Requisito y sirve como modelo para las características de los requisitos, acorde al quién, qué y por qué.

Equipo – Clase que agrupa los miembros del equipo en la aplicación. Además, es desde donde se crean, quedando así asociados.

MiembrodeEquipo – Clase para crear a cada miembro del equipo, con todos los datos necesarios.

El paquete Controlador sólo contiene una clase, la clase Controlador, llamada desde el main. En ella se crean todas las colecciones y elementos de la aplicación, además de ser a quién llama la “interfaz” para acceder al modelo, haciendo de ella el elemento intermediario de cohesión.

El paquete Vista contiene la clase main, desde donde creamos el controlador y posteriormente gestionamos la toma de datos y muestra de información.

Por último, el paquete Persistencia contiene la clase Persistencia, donde se guardan los datos en dos ficheros .csv distintos para mantener la información tras el cierre de la aplicación.

Funcionamiento.

Actualmente existen dos formas de ejecutar la aplicación desarrollada. Una de ellas es utilizar un IDE como eclipse. En ese caso simplemente compilaríamos y ejecutaríamos al momento, pudiendo ver el código paso a paso en la ejecución con el modo debugger.

La otra opción, es ejecutar el archivo Kanban.jar.

Para ello hemos de abrir la consola de comandos, ir hasta el directorio donde se encuentra y una vez ahí ejecutar el comando “java -jar kanban.jar”. También es necesario tener una JVM, dado que al tratarse de un .jar, no es un ejecutable independiente si no un archivo de Java.

Lo primero que veremos al ejecutar la aplicación es un mensaje que nos dice si queremos cargar los ficheros .csv con la información de pasadas ejecuciones. Si es la primera vez que ejecutamos la aplicación y no tenemos los archivos creados, al elegir cargar los datos dará un error ya que no puede encontrar los ficheros. Si existen, aunque no tengan información, se leerán y darán paso al menú principal. En sucesivas ejecuciones, la carga de datos se hará sin problema. Sin embargo, si se decide no cargarlos, se crearán ficheros nuevos con el mismo nombre, pisando los anteriores y borrando su contenido.

Desde el menú principal podemos seleccionar 4 opciones:

Gestión de miembros, que nos llevará al menú donde dar de alta a los miembros del equipo, eliminarlos o consultarlos.

Gestión de sprints, donde podremos añadir nuevos sprints o consultar el número de ellos que lleva la aplicación.

Gestión de tareas, donde podremos realizar las acciones referentes a las tareas,que son añadir una tarea, editar una tarea, mover una tarea del product backlog al sprint backlog actual o cambiar el estado de una tarea dentro de los posibles.

Salir, que dará fin a la ejecución del bucle del menú y por tanto del programa.

Cabe destacar de la aplicación cosas como que el cambiar una tarea de estado se hace automáticamente una vez seleccionada la opción, evitando que se introduzcan estados inexistentes o llegando a situaciones inconsistentes, como pasar de pendiente a finalizado. También, que se crea un único equipo, pero que sería fácilmente ampliable creando una estructura con los equipos ya que cada equipo tendría a sus miembros asociados.

Patrones utilizados.

Debido a problemas de tiempo, no se ha implementado ningún patrón de diseño, o al menos encontrado. No obstante, sí que hay diversos aspectos donde se podrían haber implementado.

Se podría haber utilizado el patrón “singleton” para controlar la existencia de un único product backlog, o de un único equipo, pero como la creación de ambos no está a disposición del usuario, si no que lo hace la aplicación internamente, no se ha considerado necesario.

A la hora de la carga de archivos, se hace llamando a los métodos mientras se leen los ficheros. Esto podría haberse realizado también con el patrón “estado”.

Se valoró la opción de utilizar el patrón “decorador” con la interfaz Backlog y tanto el producto backlog como el sprint backlog. Finalmente se hicieron por separado al tener muchas diferencias y no atender exactamente a lo que pide el patrón, pero se dejó la interfaz Backlog.

Así mismo, también se descartó la inclusión del patrón “fachada” por necesitar comunicación entre los 4 módulos de la aplicación. Podría haberse utilizado en la interfaz, pero al definir el método main como parte del componente vista, se requerían dependencias entre ella y el controlador.

El patrón “método fábrica” se consideró para la creación de tareas, pero no solucionaba nada con respecto a la implementación actual y por tanto no se incluyó.

Conclusión.

En esta segunda entrega se ha intentado corregir todos los defectos señalados en la primera e implementar todas las partes restantes. Finalmente, no se ha podido realizar el segundo método de persistencia debido a la falta de tiempo y conocimientos sobre el tema, pues se intentó realizar con XML pero ciertos problemas nos hicieron cambiar a CSV. También se valoró la realización mediante una base de datos SQL o con JSON, pero la falta de tiempo y experiencia en el tema nos lo impidieron una vez más.

No obstante, de éste trabajo se ha aprendido a trabajar con ficheros CSV, aplicar ingeniería del software para crear un producto software partiendo de unos requisitos (cosa que se debería haber aprendido en el 2º curso pero no se explica adecuadamente) y la valoración de aplicar ciertos patrones de diseño o no a un producto.