GRADO EN INGENIERÍA INFORMÁTICA

DESARROLLO DE SOFTWARE

UNIVERSIDAD
DE GRANADA

P3 : Pruebas software

Juan Miguel Acosta Ortega (*acostaojuanmi@correo.ugr.es*)

Jesús Pereira Sánchez (*jesuspereira@correo.ugr.es*)

David Serrano Domínguez (*davidserrano07@correo.ugr.es*)

Raúl Florentino Serra (*raulfloren@correo.ugr.es*)

4 de mayo de 2024

Índice

Objetivos

En esta práctica se van a planificar y analizar pruebas «big bang» de diferentes sistemas de la P2.

Además, a modo de depuración, se diseñarán, implementarán y ejecutarán también en esta práctica pruebas de Unidad en Dart/Flutter.

ENLACE AL REPOSITORIO DE GITHUB : Repositorio de las prácticas

1. Planificación de las pruebas del sistema

En esta tercera práctica se han de planificar y analizar pruebas del sistema de pedidos de hamburguesas que comenzamos a desarrollar en la primera práctica, y mantuvimos y mejoramos en la segunda práctica.

El primer paso será el de obtener los requisitos funcionales y puntos de vista, y requisitos no funcionales y perspectivas a partir del análisis de los patrones adoptados.

1.1. Extracción de requisitos

Nuestro sistema, como hemos presentado anteriormente, consta de la síntesis de dos patrones de diseño (Builder y Observer), estos trabajan en conjunto para poder lograr un sistema de pedidos de hamburguesas en el que el Builder se encarga de la parte de elegir recetas y cocinar las hamburguesas, y el Observer de notificar a los clientes cuando su pedido esté listo.

Los claros **Requisitos funcionales** son los siguientes:

- El sistema será capaz de construir hamburguesas de distinto tipo diferenciando una de otra por nombre, ingredientes y precio.
- El cocinero (director del patrón builder) será el encargado de la construcción de las hamburguesas (Creación de objetos complejos, patrón Builder).
- El cocinero podrá adquirir una receta (builder) y cambiarla.
- Se creará el pedido actual del cliente con toda la información necesaria (precio global del pedido y lista de hamburguesas)
- Se podrán escoger el tipo y número de hamburguesas que compondrán al pedido
- Se podrá finalizar el pedido cuando se desee, el cocinero pasará a preparar el pedido y lo cocinará en un tiempo aleatorio entre un mínimo y un máximo.
- Se notificará la finalización del pedido al usuario (Patrón observer, pues el cliente es el observer, y el cocinero es el Sujeto a observar).

Además como **Requisitos no funcionales** se podrían definir los siguientes:

- El usuario (observer) deberá ser capaz de generar su pedido haciendo click en las hamburguesas que desee. **(Usabilidad)**
- El usuario recibirá una notificación de "pedido realizado"tras un tiempo después de pedir. **(Fiabilidad)**
- El sistema debe de ser capaz de recibir varios pedidos aunque no se hayan terminado de notificar los anteriores. **(Disponibilidad)**
- El sistema ha de poder funcionar en plataformas web y móviles. **(Portabilidad)**

A continuación y con los requisitos del sistema extraídos podemos realizar el análisis, identificando las distintas condiciones de prueba y datos requeridos para las mismas.

1.2. Análisis de pruebas

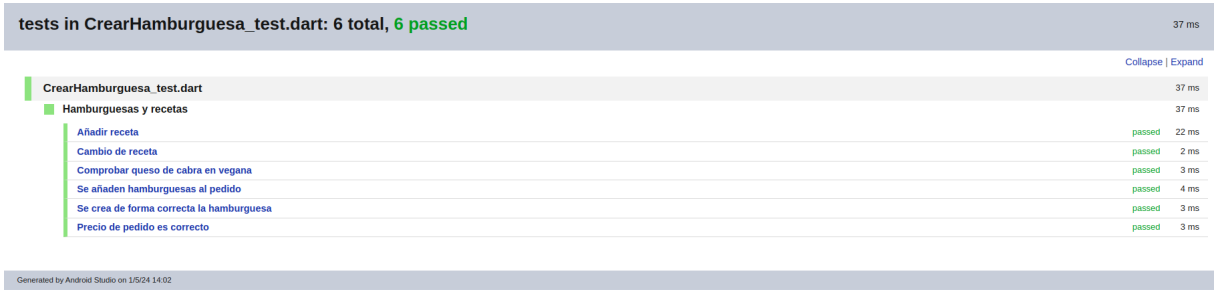
Para las pruebas unitarias hemos dividido en dos grupos bien definidos:

- Un grupo que se dedicará a hacer test del patrón Builder.
- Otro grupo que se dedicará al Observer (evaluando que también trabaja bien la combinación de ambos patrones).

Pruebas unitarias orientadas al patrón Builder:

Análisis de pruebas		
Elemento a probar	**Condiciones**	**Datos requeridos**
Director del patrón Builder	Se le puede añadir una receta	cocinero y normalBuilder
Director del patrón Builder	Se le puede cambiar de receta	cocinero, veganaBuilder y sinGlutenBuilder
Director del patrón Builder y VeganaBuilder	Se fabrica bien cada hamburguesa, en concreto la vegana no tiene queso de cabra	cocinero y veganaBuilder
Director del patrón Builder y Pedido	Comprobamos que las hamburguesas creadas por el cocinero se añaden correctamente al pedido actiual del usuario	cocinero, todos los builder diferentes y pedidoactual
Director del patrón Builder y NormalBuilder	Se comprueba si el cocinero fabrica correctamente una "Hamburguesa normalçon todos sus ingredientes.	cocinero y normalBuilder
Director del patrón Builder , todos los Builder y Pedido	Comprobamos que el precio final del pedido una vez realizado sea el esperado (cada hamburguesa tiene su precio).	cocinero, todos los builder diferentes y pedidoactual

Una vez realizamos las pruebas, exportamos los resultados:



Pruebas unitarias orientadas al patrón Observer:

Análisis de pruebas		
Elemento a probar	**Condiciones**	**Datos requeridos**
Sujeto del patrón Observer	Se le puede añadir un observador	cocinero y displayPedidos
Sujeto del patrón Observer	Se eliminan correctamente los observers	cocinero y displayPedidos
Sujeto del patrón Observer y Observers	Se notifica al observer .ªttached.ªl sujeto, en este caso sólo hay un observer	cocinero y displayPedidos
Sujeto del patrón Observer y dos Observers diferentes	Se notifican correctamente a más de un observer	cocinero y dos observers
Sujeto del patrón Observer,Pedido y Display	Se notifica correctamente el pedido actual	cocinero , displayPedido y pedidoActual
Sujeto del patrón Observer , Pedido y Observer	Comprobamos que el precio final del pedido actual sea igual que el precio del pedido de historial de pedidos.	cocinero, todos los builder diferentes , pedidoactual y displayPedidos

Una vez realizamos las pruebas, exportamos los resultados:

1.3. Diseño de pruebas

Para las pruebas unitarias hemos dividido en dos grupos bien definidos:

- Un grupo que se dedicará a hacer test del patrón Builder.
- Otro grupo que se dedicará al Observer (además de la simbiosis de los dos patrones).

Para el patrón Builder:

Diseño de pruebas			
Casos de pruebas (ordenados por prioridad)	**Entornos de prueba**	**Datos de prueba**	**Relación con las condiciones de prueba (trazabilidad)**
Construcción correcta de hamburguesa normal	Android Studio	Usamos la instancia de cocinero y dos builder del mismo tipo de hamburguesa	Cambiamos la receta del cocinero y le mandamos a cocinar. A parte, cocinamos la hamburguesa con otro builder, siguiendo los pasos esperados en el mismo orden. Al final, comparamos las 2 hamburguesas, y deben de ser idénticas.
Se añaden las hamburguesas al pedido tras cocinar	"	Un cocinero, 3 builder de distintos tipos de hamburguesas y un pedido	Cocinamos las distintas hamburguesas, y comprobamos que la longitud del pedido que tiene el cocinero coincide con las 3 hamburguesas que le hemos mandado cocinar
Añadir una receta al cocinero que creamos	"	Un cocinero y el builder que se le quiera asignar	Inicializamos el cocinero con el builder que le pasemos como argumento
Cambio de receta	"	El cocinero con un builder y el builder (distinto al inicial) que se le quiera asignar	Inicializamos el cocinero con un builder de hamburguesa vegana. Le cambiamos el builder por uno de hamburguesas sin Gluten
Comprobar queso de cabra	"	Un cocinero, un builder y un pedido	Cambiamos el builder del cocinero al de una hamburguesa vegana, mandamos a cocinar, extraemos el pedido y comprobamos que el campo de queso de cabra esté vacío
Comprobar precio del pedido correcto	"	Un cocinero, 3 builder de distintos tipos de hamburguesa y un pedido	Mandamos al cocinero a cocinar con los distintos tipos de builder que tiene (cocine 3 hamburguesas de los 3 tipos). Al final, extraemos el pedido, y el precio de este tiene que ser la suma del precio de los 3 builder de las hamburguesas

Para el patrón Observer:

Diseño de pruebas			
Casos de pruebas (ordenados por prioridad)	**Entornos de prueba**	**Datos de prueba**	**Relación con las condiciones de prueba (trazabilidad)**
Se notifica correctamente	Android Studio	Un cocinero, un builder y un observer (DisplayPedidos)	Inicializamos el cocinero con el builder, y le añadimos el observer (DisplayPedidos). Mandamos al cocinero a cocinar, y posteriormente le decimos que notifique al observador. Se espera que el historial del observer haya aumentado, con el pedido que se ha acabado
Se notifica correctamente a los distintos tipos de observadores	Android Studio. Además, se ha creado un tipo de observer para probar la correcta implementación de la interfaz de observador	Un cocinero, un builder y varios observer (DisplayPedidos y TerminalPedidos)	Inicializamos el cocinero con el builder, y le añadimos los obervers (DisplayPedidos, TerminalPedidos). Mandamos al cocinero a cocinar, y posteriormente le decimos que notifique al observador. Se espera que el historial de los observers haya aumentado, con los pedidos realizados.
Añadir observador al cocinero	"	Cocinero y observador	Se le añade el observador al cocinero
El pedido notificado es correcto	"	Cocinero, builder de receta, observer y pedido	Mandamos el cocinero a cocinar. Extraemos el pedido. Notificamos a los observadores. Comprobamos que el pedido del historial del observer es el mismo que el pedido que tenía el cocinero antes de notificar (cuando notifica, borra el pedido)
Eliminamos el observador del cocinero	"	Un cocinero y observador	Se elimina el observador del cocinero
El precio del pedido notificado es el precio correcto	"	Un cocinero, 3 builder de distintos tipos de hamburguesa, un pedido y un observer	Mandamos al cocinero a cocinar con los distintos tipos de builder que tiene (cocine 3 hamburguesas de los 3 tipos). Al final, extraemos el pedido. Notificamos a los observadores. Comparamos el precio del pedido de uno del los observadores con el precio del pedido de antes de notificar (deben de coincidir)

Todos estos tests se diseñan y desarrollan con la finalidad de poder probar todo el modelo al completo del sistema, y comprobar que efectivamente cada componente se desenvuelve correctamente tanto por separado como en conjunto.

2. Pruebas de componentes (widgets)

Por último, y por curiosidad, queríamos probar haciendo tres tests sencillos sobre los widgets de Flutter.

Los tests asíncronos de los widgets de Flutter son tests que pueden manejar tareas asíncronas dentro de los widgets, como la construcción de widgets diferida. Por ejemplo, pumpWidget se utiliza para construir un widget y luego actualizarlo con pump para simular el paso del tiempo o eventos asíncronos, como la animación o la recuperación de datos. Esto permite probar casos de uso complejos que involucran interacciones asíncronas dentro de la interfaz de usuario.

En resumen, estos tests nos permiten verificar el comportamiento de nuestra aplicación desde la perspectiva del usuario, asegurándonos de que la navegación funcione correctamente y de que las interacciones del usuario produzcan los resultados esperados, como el aumento del contador del carrito en este caso.

Para este tipo de pruebas se han realizado estas sencillas comprobacines:

- Los dos primeros test comprueban que haya una navegación efectiva desde una pantalla a otra desde dos botones distintos.
- El tercero comprueba que se incrementa el contador del carrito al añadir diferentes hamburguesas al pedido.

Estos test fueron pasados con éxito tras arreglar un componente de la pantalla Menu, es decir, si un widget no es completamente responsive no pasará los test, es por eso que se ha de pensar en este estilo de diseño desde el principio del desarrollo de la UI incluso pensando en los test futuros, no sólo en la experiencia de usuario.
Este arreglo consistió en otorgarle el hijo Expanded (equivalente a display: flex; flex-direction:column;) al widget Column donde van los botones de las hamburguesas.

Estos son los tests pasados: