Reflexiones del proyecto Ramenautos

¿Qué cosas salieron bien y qué cosas salieron mal? ¿Qué decisiones resultaron acertadas y qué decisiones fueron problemáticas?

Para empezar, creemos que es de destacar que la división de los requerimientos en función de los usuarios (administrador, cliente y empleado), así como el modelo de control delgado funcionó muy bien. De esta manera, cada uno de nosotros se encargaba de los requerimientos relativos a su usuario asignado de manera independiente y creando sus propias interfaces. Así, la estructura común eran solamente los data holders (modelo), las clases encargadas de la persistencia, y la ventana principal de la interfaz general. De esta forma, se logró reducir el acoplamiento bastante, y que una falla en algún requerimiento no implicara la caída total del sistema. Además, se sigue especialmente el principio de Interface segregation, y se promovía la mayor autonomía de cada miembro del grupo.

Pensamos sin embargo, que fue especialmente problemático manejar la persistencia en solo tres clases. Esto hizo que estas tres clases quedaran de aproximadamente 500 líneas de código, lo que no solo dificultaba encontrar el elemento preciso a desarrollar o editar sino que además hacía que la frontera de estas clases fuera muy grande (ya que estaban muy relacionadas con muchos otros elementos), lo que aumentaba mucho el acoplamiento y la interdependencia de clases. De esta forma, el programa depende mucho de la implementación y excelente funcionamiento de estas clases, no siguiendo el principio de Single Responsability.

En la primera entrega salió mal, que no se probó la creación de reservas y por eso el programa falló en esa dicha primera entrega, pero esto fue por un asunto de fuerza mayor.

¿Qué tipo de problemas tuvieron durante el desarrollo de los proyectos y a qué se debieron? En este último punto, sería conveniente discutir aspectos como los problemas con las estimaciones del trabajo necesario debido al desconocimiento de la tecnología, los problemas realizando el análisis del dominio, o la dificultad de diseñar en un entorno incierto.

Generales: En general al principio hubieron muchos problemas con la tecnología, no sabíamos usar bien Java ni Eclipse, lo que ralentizaba mucho el trabajo. Por ejemplo, no sabíamos debugear, tuvimos problemas con el merge en GitHub y para las clases del modelo que consistían principalmente en getters y setters, perdimos mucho tiempo haciéndolas manualmente, pues no sabíamos que Eclipse podía poner automáticamente estos métodos.

Por otra parte, para realizar la carga y descarga de los archivos, tomó mucho tiempo realizarlo, ya que había diferencias entre la forma en que se escribían los archivos y como el programa los reconocía. Esto causó demoras en la finalización del proyecto, debido a que estuvimos solucionando por varias horas un problema que se podía haber evitado desde el principio. Asimismo, crear un pdf consumió mucho tiempo debido al desconocimiento en cómo descargar una nueva librería y como esta se utilizaba.

ESPECÍFICOS:

Parte del cliente: Además de los problemas relacionados con el desconocimiento de la tecnología, como el hecho de haber usado para muchas de las validaciones en las reservas un == en vez del método isEqual (lo que producía resultados indeseados que pasaron inadvertidos para la primera entrega), también hubieron problemas relacionados con diseñar en un entorno incierto. El enunciado era muy poco claro con algunos requerimientos funcionales, por lo que se tenían que añadir pautas e interpretaciones adicionales para cumplir, lo que al principio fue difícil, pues no nos habíamos enfrentado nunca a esto. Por ejemplo, para la primera entrega, se decía que se debía procurar con las reservas, pero al mismo tiempo que un empleado podía poner carros en mantenimiento sin restricciones adicionales. Si esto fuera libre, el empleado podría poner los

carros en mantenimiento por plazos tan largos lo que en el caso extremo imposibilitaría que se cumpla con las reservas, pues estas se hacen antes. Así, se optó por ponerle un tope de 2 días al periodo de mantenimiento y asegurar que un carro no pueda reservarse para un periodo que inicie en menos de 2 días de la devolución de la reserva anterior. Otro ejemplo es el caso de la consulta de disponibilidad de la tercera entrega, no quedaba claro como determinar si un carro iba a estar disponible en una fecha en una sede o no, dado que el administrador podía cambiar los carros de sede a su antojo y además los alquileres podían extender las reservas. Así, se optó por contar solamente un carro disponible teniendo en cuenta solo su estado de alquiler, que es cuando la trayectoria del carro ya es segura.

Parte Administrador:

Esta parte era realmente sencilla, los problemas vienen de lo mismo que se ha dicho con respecto a la clase BaseDatos; el alto acoplamiento, es molesto recurrir a esta clase, se hace muy dependiente. Como solución, pudimos repartir los métodos y atributos de BaseDatos en las respectivas clases del modelo dando responsabilidad a cada una de estas clases en lugar de todo a una sola.
Hubo problemas en la entrega 2 con respecto al diagrama debido a que cambiamos el archivo txt de carros para nuestros requisitos y eso hacía que el diagrama no funcionara en dichos casos, por lo que en algunos casos veía que necesitaba información pero alguno otro integrante (samuel) borraba los datos también para su requisito pero como el diagrama necesitaba varios datos los necesitaba completo . Era una especie de tire y afloje que a decir verdad no se soluciono del todo.