Universidad ORT Uruguay

Obligatorio

Ingeniería de Software Ágil 2

Agustina Disiot 221025

Beate Lidstrom 292178

Iván Monjardin 239850

Francisco Rossi 219401

ISA2/Obligatorio (github.com)

Rama: entrega-3

2022

Índice:

Introducción

En este informe se va a encontrar la explicación del trabajo realizado para la tercera entrega del obligatorio de Ingeniería de Ágil 2.

Se planificó realizar lo pedido en la letra para la tercera entrega lo cual cumplimos en hacer. En las distintas secciones se incluyen comentarios detallando los resultados, justificaciones y explicación de los procesos.

Los subtítulos con números al inicio corresponden a los puntos pedidos en la Definition of Done correspondiente.

1. Implementación del frontend:

Antes de explicar ambas funcionalidades de alta y baja de un punto de carga cabe aclarar que se decidió hacer la implementación de una tercera funcionalidad, obtener todos los puntos de carga del sistema. Esta funcionalidad se realizó siguiendo el marco de BDD, con SpecFlow para el backend y Selenium para el frontend. Aunque se entiende que se agregó complejidad a la entrega, se decidió realizarlo de esta manera, ya que creemos importante esta funcionalidad para darle un mejor sentido al posible uso que el usuario final puede llegar a darle al sitio web.

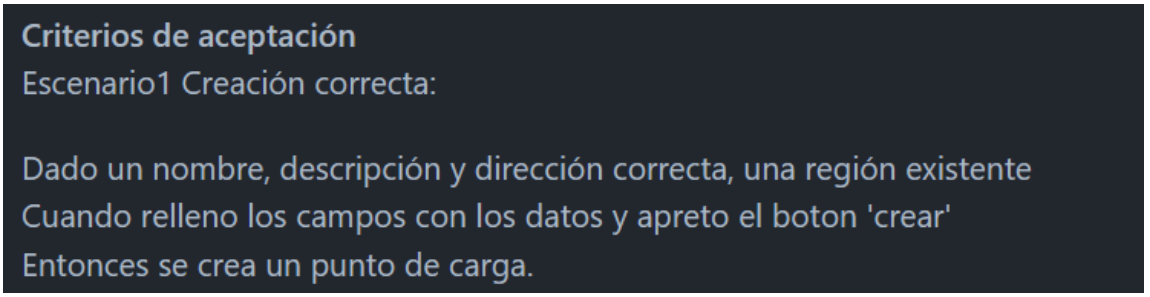
Para las 3 funcionalidades, se decidió comenzar por el backend, siguiendo con el backend siguiendo el marco de BDD para todos los casos.

A continuación se puede observar una imagen que representa las funcionalidades creadas desde cero:

2. Selenium:

Se utilizó la herramienta Selenium con el framework Chai para automatizar las pruebas funcionales del frontend siguiendo el marco de BDD.

Esta herramienta permite poder tomar escenarios de la User Story correspondiente, que fueron definidos en la etapa de los Requerimientos. Por ejemplo al comprar el mismo escenario que se encuentra en la User Story en GitHub y la que se utilizó en Selenium:




```
Scenario: Create a valid charging point
  Given I view the "http://localhost:4200/explore/create-charging-point"
  When I type "Ancap" in the field "name"
  When I type "Desc" in the field "description"
  When I type "maldonado" in the field "address"
  When I choose "Región Metropolitana" in the combo "region"
  When I click on button Crear
  When I wait for 3000 ms
  Then the page should show a message saying "Exito!"
```

Se pueden observar que existen diferencias, pero en términos generales se está testeando exactamente lo mismo. En Selenium decidimos poner la información más específica que en GitHub que se hizo de una forma más general. También se debe de incluir exactamente la dirección del sitio web donde se quiere realizar el testing automático, ya que es algo que requiere la herramienta para poder probar los casos correspondientes.

El proceso que siguió fue el siguiente. Luego de definir los requerimientos de la User Story, se creaban los casos de prueba a través de estos escenarios previamente mencionados, los cuáles se mapean a un archivo .step que ejecuta el test de forma automática. En este momento todos los escenarios fallan, ya que no existe él código de la funcionalidad. El desarrollador empieza a escribir el código para avanzar en la User Story y una vez que lo termina se prueba. La idea es llegar todo el sistema en conjunto, front y back para ver que la integración de la nueva funcionalidad está completa y pasa los test automáticos, si esto no ocurre, se debe recurrir al refactor tanto de detalles de las pruebas o del código mismo.

Lecciones aprendidas y Errores

En esta entrega aprendimos mucho y tuvimos varios contratiempos. Fue la primera vez que utilizamos Selenium y herramientas similares para realizar test funcionales y tests de integración.

Las nuevas herramientas trajeron problemas de instalación y de utilización. Pudimos solucionar problemas que tuvimos con la instalación inicial de Angular pero la instalación y configuración de las nuevas herramientas nos llevó más tiempo de lo esperado.

Además la integración de nuevas GitHub actions también fueron más costosas de lo esperado principalmente con la dificultad de probarlas. A veces correr los mismo comandos localmente funcionaba pero en GitHub Actions no, debuggear requería realizar el commit y esperar a que se ejecute la GitHub Action para ver el resultado (lo cual generaba bastantes “Failed” Workflow).

3. Sobre la revisión (review con el product owner)

Se realizaron 2 reviews con el product owner siendo algunos de los integrantes que no participó en algunas de las User Stories.

El vídeo se encuentra en el siguiente link: https://youtu.be/GVZlPGmA1pg

Proceso

Para esta etapa, se decidió utilizar una herramienta llamada Selenium IDE. La idea es básicamente igual a la de Selenium previamente mencionada, con el agregado de grabar el testing automático del front end. Esto permite que quede un registro de la funcionalidad nueva andando correctamente, y poder observar concretamente cómo se comporta el sistema al momento de la grabación.

La idea que utilizamos para que no sea el mismo desarrollador el que haga de product owner en la review de su funcionalidad, es ir rotando el integrante del grupo que no participó en el desarrollo de la misma. Por ejemplo si alguno desarrolló el obtener todos los puntos de carga, podía hacer de PO para la baja de puntos de carga si no participó en ellas. El PO debía leer los requerimientos de la User Story, previo a correr la prueba con Selenium IDE para poder tener una idea más clara de que debería realizarse en la prueba y si cumple o no con lo definido previamente.

En las grabaciones se pueden observar las 3 nuevas funcionalidades que creamos de forma que se integra el front con el back, obteniendo y actualizando la información de la base de datos. Las grabaciones se encuentran en el repositorio en la carpeta Obligatorio/TestSelenium

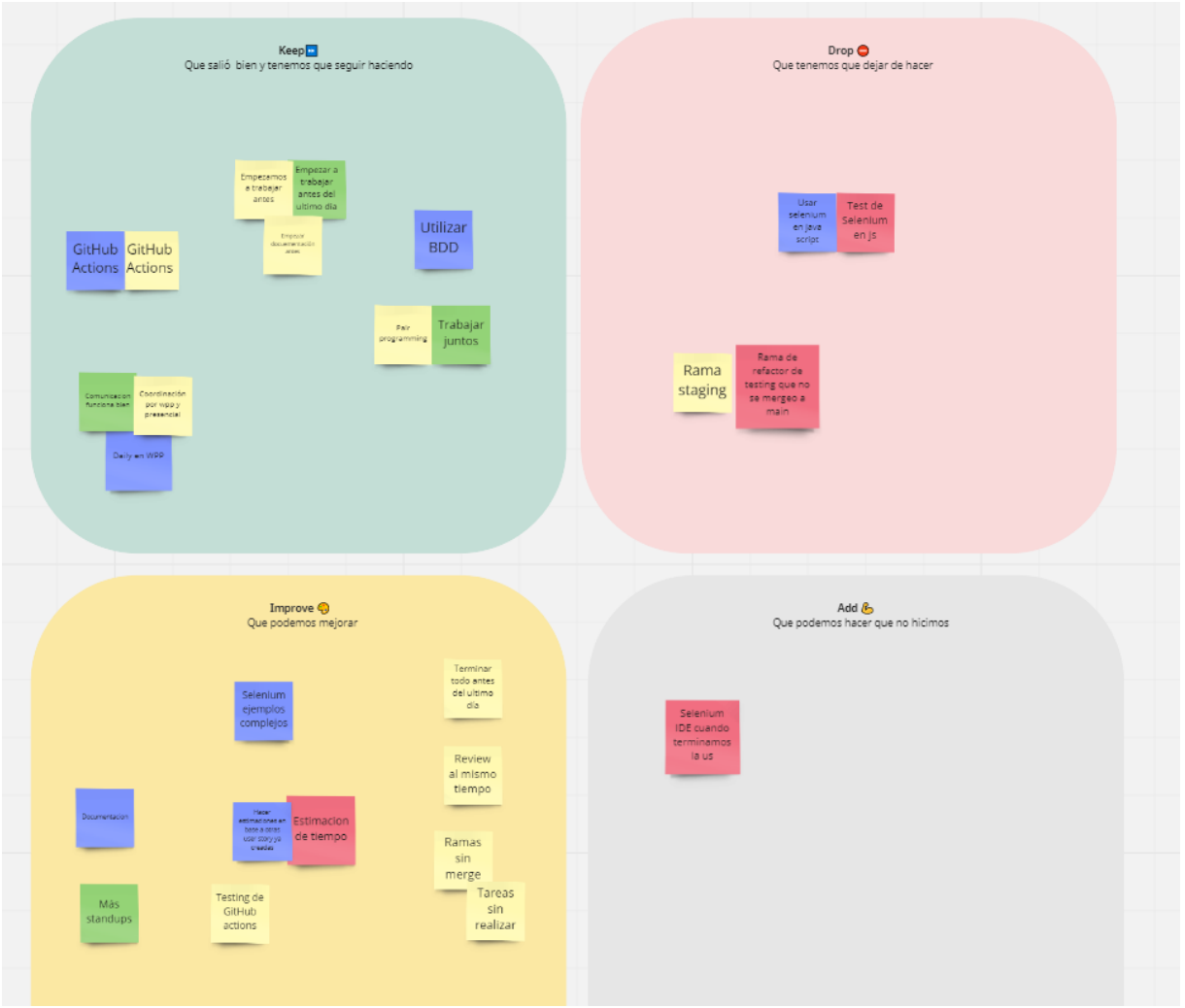
Retrospectiva

Al final de las dos semanas se realizó una retrospectiva con todo el equipo. La idea fue revisar qué cosas podemos mejorar para la siguiente entrega.

La sesión fue grabada y está disponible en el siguiente link:

https://www.youtube.com/watch?v=88fRnfk8IP0

Se utilizó el método DAKI (Drop, Add, Keep, Improve) en un tablero Miro el cual podemos ver a continuación



Se discutieron varios temas en la retrospectiva como se puede observar en el video, donde cada uno propuso ideas de tópicos para discutir luego de 7 minutos.

Algunos resultados a destacar:

- Continuar con la posibilidad de realizar pair programing para el que lo desea y coinciden los horarios. *
- Utilizar BDD para simular lo que es una entrega a un cliente real: BDD tanto con SpecFlow y, en esta iteración, con Selenium nos permiten presentar los tests de forma que simule el comportamiento como un cliente real o describiria y mediante estas instrucciones programas dicho comportamiento.
- Buena comunicación, seguir haciendo stand ups por WhatsApp. *
- Estamos usando GitHub actions. Con este podemos controlar los build de backend y frontend automáticamente al hacer push o merge a main
- Estas dos semanas empezamos más temprano con el proyecto que ha funcionado muy bien. Si bien tuvimos problemas con las herramientas de Selenium, comenzar antes fue de utilidad para que las dificultades no se estancaran al final de la iteración.
- Coincidimos que dejar de usar JavaScript o node para los tests de Selenium ya que estos tests son independientes del lenguaje, y al investigar sobre como hacer los tests, se encontró más documentación o soluciones en otros lenguajes como Java o .NET.
- Los ejemplos de Selenium son muy complejos. Aquí lo que se quiso expresar es que los tests de BDD son concretos y testean funcionalidades solo en algunos casos en lugar de ser genéricos.
- Queremos mejorar nuestra estimación de duración y esfuerzo de los issues.
- Ahora hacemos ramas sin merge, queremos hacer más merges al main para trabajar más como trunk based qué git flow.
- Otros.

También nos parece importante destacar algunas acciones que se tomaron en función de la retrospectiva anterior, los especificados de los puntos arriba están marcados con “*”.

Estándar de versionado

Se realizaron cambios al estándar de versionado en comparación a lo definido anteriormente:

- En función de lo preguntado en clase y lo definido por el equipo se decidió sacar la rama de staging ya que nuestro 'staging' debería ser la rama main.

Deploy a ambiente de staging

Se realizó un deploy manual de la aplicación a la nube para poder realizar la review en un ambiente de staging y no corriendo nosotros la aplicación.

Se intentó automatizar el despliegue de algunas partes pero se terminó priorizando otras tareas.

El link al ambiente de staging es el siguiente: http://169.51.203.232:32627/, aunque probablemente cambie porque el ambiente es un trial y se vence a unos días de la entrega 3.

Análisis de Métricas

Una de las métricas que tomamos en cuenta es el Work In Progress (WIP), el cual indica cuántas tarjetas en el tablero están en el mismo momento en "Doing" haciendo referencia a cualquier columna que implique estar en proceso de trabajo para nuestro caso.
Lo que queríamos lograr y logramos fue que el WIP nunca sea mayor a 4, ya que contamos con cuatro integrantes en el equipo.
Tomando en cuenta que el WIP solo aplica para las User Stories (e.g. Bugs y funcionalidades nuevas), nunca fue mayor a 4. Como hicimos

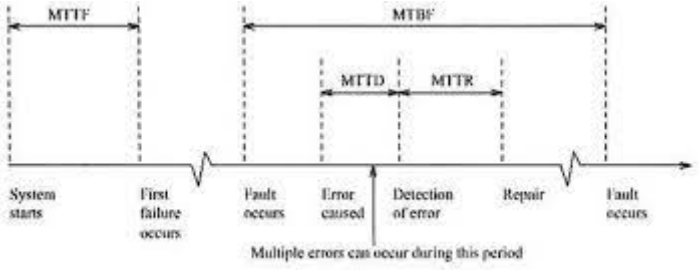
Otra métricas es el Deployment Frequency que en esta caso, como no deployeamos a producción, lo tomamos como la cantidad de veces que hacemos el "deploy" a staging. Cómo en esta entrega usamos un ambiente de staging se realizaron varios deploy a staging, principalmente cuando comenzaron las dos semanas y sobre el final, el Deployment frequency fue de 3 cada dos semanas.

Lead time vs Cycle time
En pocas palabras, el tiempo de ciclo mide el tiempo que le toma a un equipo fabricar un producto, mientras que el tiempo de entrega mide el tiempo entre el pedido del cliente y el cumplimiento del pedido. El tiempo de entrega siempre es más largo que el tiempo del ciclo porque el tiempo del ciclo se ajusta a la línea de tiempo del tiempo de entrega.

El lead time esta entrega lo mediremos para las funcionalidades que se pedían en la Definition of Done: Alta y Baja de un punto de carga, siendo que el lead time es de dos semanas considerando que las funcionalidades se pidieron al comienzo de la iteración 3. Sin embargo

el Cycle time de la alta y baja, desde que se pasa la tarjeta a Requerimientos hasta que llega a Done fue de 10 horas (coinciden para ambas US).

MTTR es la métrica Mean Time To Repair (Tiempo estimado o promedio para reparar). En esta entrega no arreglamos fallas, entonces usamos la entrega anterior para mostrar cómo lo calculamos. MTTR es el tiempo entre encontrar el bug y cuando está arreglado. Arreglamos dos bugs, uno tardó 2h y uno 6h. Entonces MTTR es (2+6)/2= 4h.



Registro de Esfuerzo

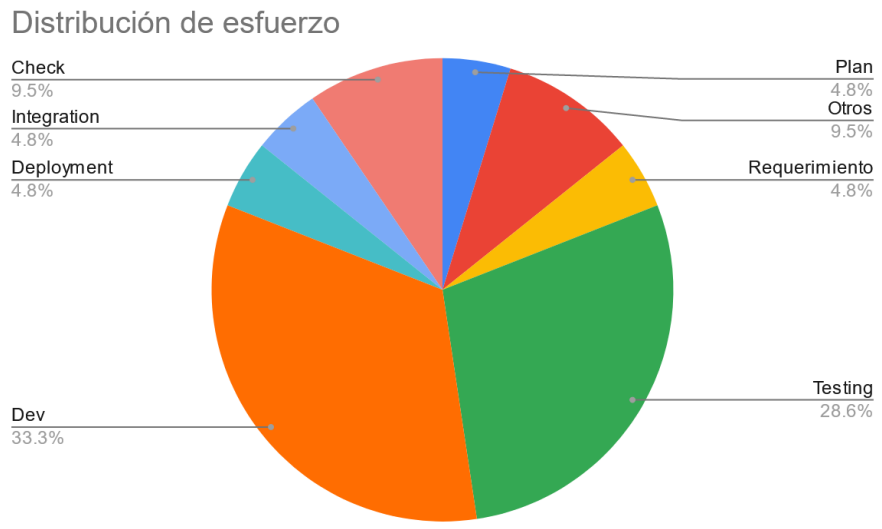
Como se comentó, se registró el esfuerzo en un Excel:

Tercera entrega						
	Duración (horas)	Personas	Estimación (esfuerzo)	Esfuerzo (horasXPersona)	Diferencia real - estimado	Tipo de tarea
Lectura de definition of Done 3 y creación de tareas	0.6	4	0.5	2.4	1.9	Plan
Documentacion siguiendo Definition of Done	1	4		4	4	
Instalar Selenium	1	4	0.5	4	3.5	Otros
Obtener todos los charging points (Requerimientos)	0.16	2	0.16	0.32	0.16	Requerimiento
Obtener todos los charging points backend (TCI)	1.6	2	0.5	3.2	2.7	Testing
Obtener todos los charging points backend (Dev)	1	2	1	2	1	Dev
Obtener todos los charging points backend (Acceptance)	0.08	2	0.08	0.16	0.08	Testing
Obtener todos los charging points frontend (TCI)	0.75	1	0.5	0.75	0.25	Testing
Obtener todos los charging points Front (Dev)	2.5	1	2	2.5	0.5	Dev
Obtener todos los charging points frontend (Acceptance)	0.033	1	0.033	0.033	0	Testing
Obtener todos los charging points frontend (Refactor)	5	1	0.08	5	4.92	Dev
Integration testing Alta y baja	0.5	2	0.033	1	0.967	Testing
Automatización del build del front	0.6	1	1	0.6	-0.4	
Deploy a la nube	6	1	4	6	2	Deployment
GitHub Action code Coverage	4	1	1.5	4	2.5	
Crear charging point Front (TCI)	1	1	0.8	1	0.2	Testing
Crear charging point Front (Dev)	3	1	1	3	2	Dev
Refactor con Selenium Crear charging point	1	1	1	1	0	Dev
Refactor Selenium Crear y Eliminar charging point	5	3	1	15	14	Dev
Front end baja de punto de carga	4	2	1.5	8	6.5	Dev
Aprender lenguaje/ over the shoulder programming con los otros	4	1		4	4	Otros
Selenium IDE	2.5	2		5	5	Integration
Retrospectiva	0.5	4		2	2	Check
Reviews				0	0	Check
			Total	74.963	57.777	

Nota: Las estimaciones también están en la pipeline definida en GitHub.

Nota: Over the shoulder programing es un concepto de mirar a alguien programar para aprender y para poder entender el proyecto y ayudar a evitar errores.

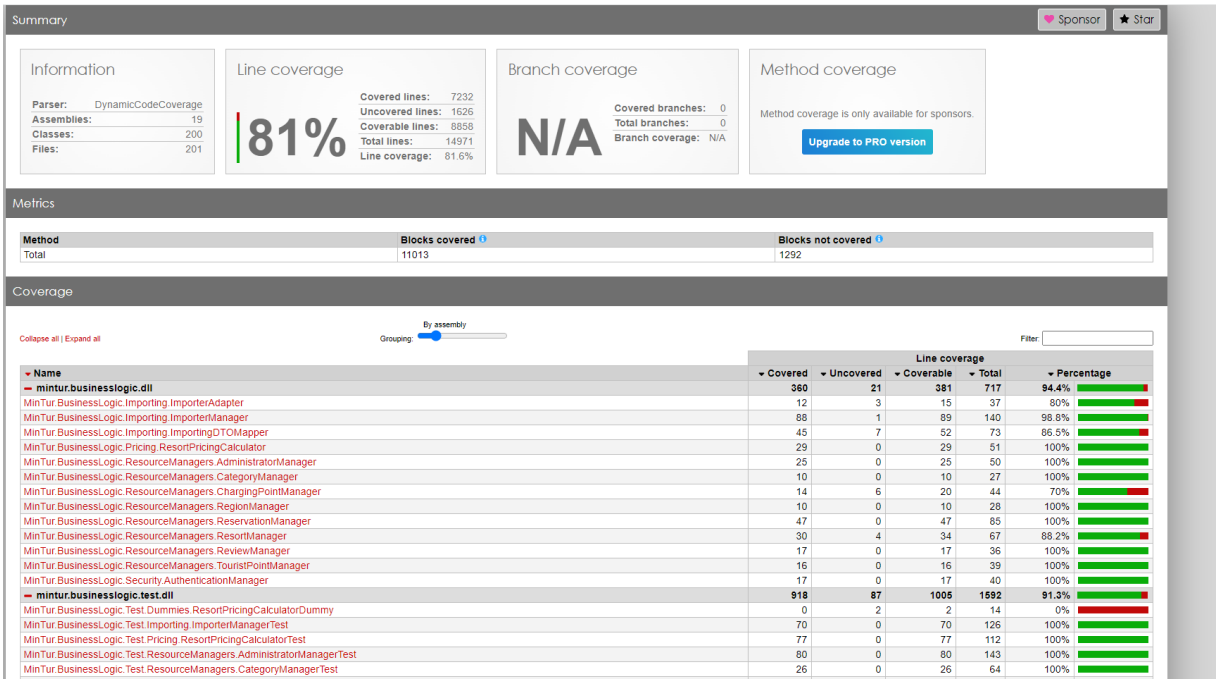
También se realizó una distribución de esfuerzo según lo recomendado en clase:



GitHub Actions

Se hicieron las siguientes modificaciones usando GitHub Actions:

- Se automatizó el proceso de Build del front end y se sube el resultado como Artefacto de la workflow
- Se arregló problema con Build del SpecFlow (se requirió agregar “dependencias” que no eran necesarias localmente pero si en el GitHub Actions, ej. using System.Thread.Task)
- Code Coverage - Se sube como artefacto del workflow, es un html que se ve de la siguiente manera:



Summary

Sponsor ★ Star

Information

Parser:	DynamicCodeCoverage
Assemblies:	19
Classes:	200
Files:	201

Line coverage

81%

Covered lines:	7232
Uncovered lines:	1626
Coverable lines:	8858
Total lines:	14971
Line coverage:	81.6%

Branch coverage

N/A

Covered branches:	0
Total branches:	0
Branch coverage:	N/A

Method coverage

Method coverage is only available for sponsors.

Upgrade to PRO version

Metrics

Method	Blocks covered	Blocks not covered
Total	11013	1292

Coverage

Collapse all | Expand all

By assembly

Grouping:

Filter:

	Line coverage				
Name	Covered	Uncovered	Coverable	Total	Percentage
mintur.businesslogic.dll	**360**	**21**	**381**	**717**	**94.4%**
MinTur.BusinessLogic.Importing.ImporterAdapter	12	3	15	37	80%
MinTur.BusinessLogic.Importing.ImporterManager	88	1	89	140	98.8%
MinTur.BusinessLogic.Importing.ImportingDTOMapper	45	7	52	73	86.5%
MinTur.BusinessLogic.Pricing.ResortPricingCalculator	29	0	29	51	100%
MinTur.BusinessLogic.ResourceManagers.AdministratorManager	25	0	25	50	100%
MinTur.BusinessLogic.ResourceManagers.CategoryManager	10	0	10	27	100%
MinTur.BusinessLogic.ResourceManagers.ChargingPointManager	14	6	20	44	70%
MinTur.BusinessLogic.ResourceManagers.RegionManager	10	0	10	28	100%
MinTur.BusinessLogic.ResourceManagers.ReservationManager	47	0	47	85	100%
MinTur.BusinessLogic.ResourceManagers.ResortManager	30	4	34	67	88.2%
MinTur.BusinessLogic.ResourceManagers.ReviewManager	17	0	17	36	100%
MinTur.BusinessLogic.ResourceManagers.TouristPointManager	16	0	16	39	100%
MinTur.BusinessLogic.Security.AuthenticationManager	17	0	17	40	100%
mintur.businesslogic.test.dll	**918**	**87**	**1005**	**1592**	**91.3%**
MinTur.BusinessLogic.Test.Dummies.ResortPricingCalculatorDummy	0	2	2	14	0%
MinTur.BusinessLogic.Test.Importing.ImporterManagerTest	70	0	70	126	100%
MinTur.BusinessLogic.Test.Pricing.ResortPricingCalculatorTest	77	0	77	112	100%
MinTur.BusinessLogic.Test.ResourceManagers.AdministratorManagerTest	80	0	80	143	100%
MinTur.BusinessLogic.Test.ResourceManagers.CategoryManagerTest	26	0	26	64	100%

Objetivos

Como se recomendó en clase, para esta entrega se decidió utilizar OKR para definir los objetivos del proyecto y en el **informe final se escribirán los Key Results**. Nos basamos en los objetivos de la letra del obligatorio, agregando y modificando algunos:

- Objetivo: El WIP sea menor o igual a 4
- Objetivo: Analizar deuda técnica del proyecto
- Objetivo: Crear un pipeline con evento y acciones
- Objetivo: Aplicar un marco de gestión ágil
- Objetivo: Implementar un repositorio y procedimientos de versionado.
- Objetivo: Crear un pipeline con eventos y acciones.
- Objetivo: Generar escenarios de testing desde la perspectiva del usuario.
- Objetivo: Automatizar el testing funcional o de caja negra.
- Objetivo: Reflexionar sobre DevOps.