Entrega 1

Ingeniería de software ágil 2

Integrantes:

Rodrigo Haiache - N° 197344
Gabriel Lutz - N° 173507
Marcos Novelli - N° 201663
Martina Praderio - N° 242494

N7A - 2022

Universidad ORT del Uruguay

Repositorio de GitHub
https://github.com/ORT-ISA2-2022S2/obligatorio-equipo-6

Índice

Introducción **3**

Proceso de Ingeniería **3**

Tablero Kanban 3

Métricas 4

Daily meeting 5

Retrospective meeting 5

Repositorio **6**

Estructura 6

Versionado 6

Elementos que lo componen 7

Análisis de deuda técnica **10**

Testing Exploratorio (Experiencia e interfaz de usuario) 10

Testing Backend 10

Testing Frontend 11

Issues (descripción y clasificación) 12

Resumen de Issues 13

Retrospectiva basada en DAKI **18**

Resultados obtenidos en el periodo 19

Dificultades encontradas y formas de solución 19

Lecciones aprendidas y mejoras en el proceso 19

Detalle de registro de esfuerzo 20

Introducción

Nuestro primer paso fue realizar una primera reunión por “Google Meet” para adentrarnos al obligatorio, entender bien los requerimientos para así poder planificar y organizar nuestra modalidad de trabajo, horarios de reunión y los pasos a seguir para poder así completar la entrega.

Proceso de Ingeniería

Tablero Kanban

Se decidió utilizar la aplicación Trello para la implementación de nuestro tablero Kanban.
Crearemos un tablero diferente por cada entrega a realizar. Para esta entrega (entrega 1) se utilizará un Tablero Ágil con las siguientes columnas:

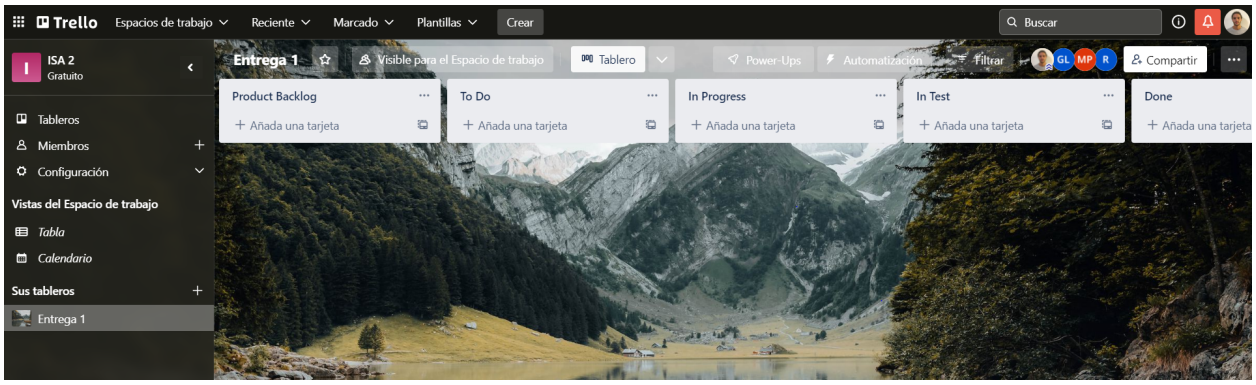
- **Product Backlog:** cuando creamos un nuevo ticket, lo hacemos en esta sección.

- **To do**: cuando un ticket que está en el Product Backlog contiene todos los detalles necesarios como para poder ser realizado, este se mueve al sección de To Do.

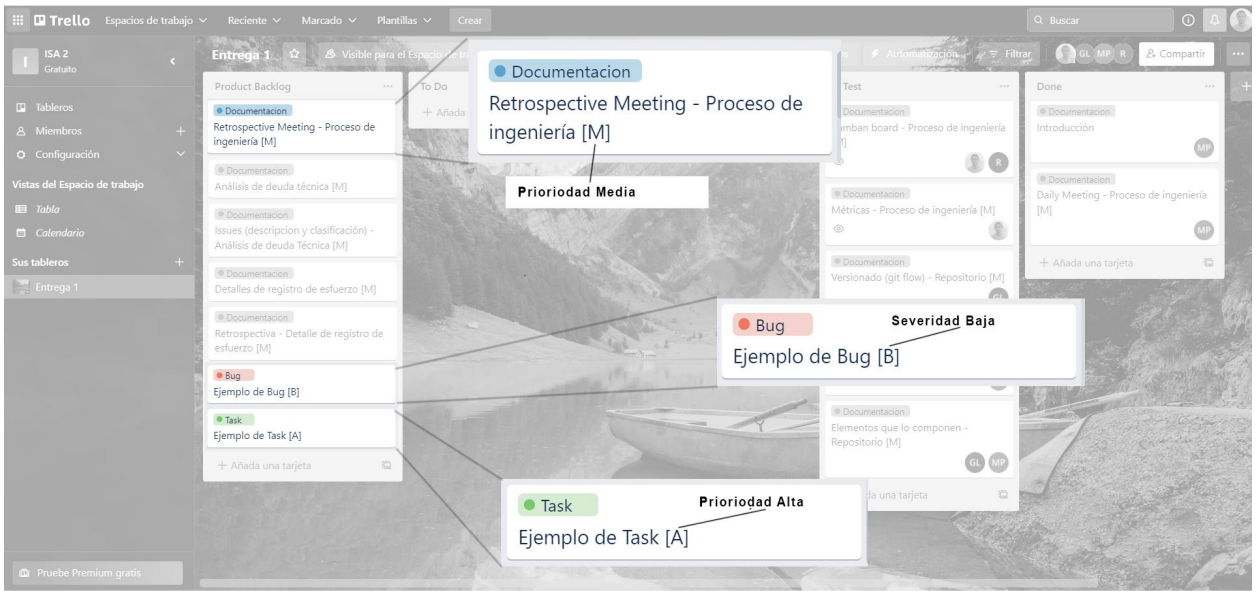
- **In progress**: en esta sección se encuentran todas aquellos tickets que alguien está desarrollando en ese preciso momento.

- **In test**: luego de terminar el desarrollo de un ticket, este se mueve a la sección de In Test. Acá es donde se debe testear si lo implementado funciona correctamente. En el caso de que así sea, el tester mueve el ticket a Done, en el caso de que encuentre errores sobre este ticket desarrollado, escribe los errores en los comentarios del ticket, vuelve a mover el ticket a In Progress y le avisa a la persona que lo implementa que encontró fallas.

- **Done**: esta sección llevará el registro de todas los tickets finalizados.



Presentación de métricas en el Tablero:



Métricas

Las métricas que utilizaremos para este trabajo basado en una metodología Kanban son las siguientes:

- **Lead time**: tiempo que transcurre desde que el un ticket se crea en el backlog hasta que este ticket se mueva a done.
- **Cycle time**: tiempo que transcurre desde que un ticket sale desde el backlog hasta que se mueva a done.
- **WIP (Work In Progress)**: cantidad de tickets en los que el equipo está trabajando al mismo tiempo. Kanban, manifiesta que se deberá limitar este WIP

tomando como cota superior la cantidad de miembros del equipo. Para nuestro caso, el WIP nunca podrá ser mayor a 4.
- **Throughput**: la cantidad de tickets finalizados en determinado período de tiempo. Para este trabajo, tomaremos este período de tiempo como un ciclo de entrega.
- **Prioridad**: para nuestro trabajo decidimos que cada integrante que cree un ticket con la etiqueta “Task” o “Documentación” sea el que le asigne la métrica de prioridad. Esta puede ser “Alta”, “Baja” o “Media”.
- **Severidad**: para nuestro trabajo decidimos que cada integrante que cree un ticket con la etiqueta “Bug” sea el que le asigne la métrica de severidad. Esta puede ser “Alta”, “Baja” o “Media”.

Daily meeting

Para esta entrega debido a sus requerimientos, decidimos no proseguir el evento Daily Meeting como Scrum, sino que nos reunimos 3 veces por semana para trabajar en requerimientos juntos, como a su vez para informarnos y ponernos al día de lo trabajado cada uno por separado.

Retrospective meeting

Esta reunión se hará al terminar la entrega 1 para compartir nuestras opiniones sobre nuestra metodología de trabajo, ponerlas sobre la mesa y discutirlas para poder progresar en nuestro trabajo como equipo para la siguiente entrega.

Como guía, utilizamos la actividad de análisis DAKI (Drop, Add, Keep, Improve) con la herramienta Miro. Drop refiere a las cosas que consideramos que estamos haciendo mal y deberíamos dejar de hacerlas; Add son cosas que deberíamos comenzar a hacer; Keep se refiere a lo que consideramos que estamos haciendo bien y deberíamos continuar haciéndolo; Improve es lo que estamos haciendo pero podríamos mejorar.

Primero, tendremos un tiempo como para pensar y escribir tarjetitas en el tablero del DAKI. Luego iremos tarjeta por tarjeta conversando el tema tratado y el responsable de escribirla tiene la oportunidad de explicar detalles de la misma. Luego, con vista a la próxima entrega, analizamos los resultados para poder mejorar el proceso de trabajo para la entrega 2.

Repositorio

Creamos un repositorio en GitHub, incluyendo a los cuatro participantes del grupo.

Estructura

En la raíz del repositorio vamos a incluir el material de obligatorio provisto, como así una carpeta con cada entrega. Dentro de cada entrega vamos a tener subpartes según sea necesario para el entendimiento de las mismas. A su vez para separar lo hecho en cada entrega realizaremos un release al terminar cada una de ellas.

Versionado

Consideramos que es muy importante elegir una buena estrategia, ya que la manera en que integramos el código puede llegar a impactar en la productividad del equipo. Al comenzar a analizar qué estrategia utilizar, nos planteamos Trunk-Based Development, ya que la misma fue dictada en el ámbito del curso y nos permitía tener una serie de ventajas a aprovechar.

Dentro de estas ventajas, observamos que siempre podíamos trabajar sobre el código más reciente, evitarnos grandes conflictos y merges extras (lo que se traduce en gasto de tiempo), releases por demanda, dado que el código en el trunk siempre está listo y en buen estado (deberíamos poder hacer release en cualquier momento), lo que la conjunción de estos beneficios se traducía en que el equipo se vuelva más ágil y eficiente a la hora de entregar código.

A primera vista, esta estrategia parecía ser la ideal para aplicarla, pero ahondando más en el tema nos dimos cuenta que al no tener una frecuencia de release alta y si el equipo se encuentra bien establecido con tiempos y fechas de releases bien definidas, la simplicidad de Gitflow nos era suficiente.

A su vez, como estamos en un entorno académico y seguimos aprendiendo, Trunk-Based Development necesita que los desarrolladores sean responsables de su código y hacer nuestro mayor esfuerzo para subir código de calidad y de esta manera no introducir bugs, así como también automatizar desde unit tests hasta la infraestructura para tener una continua integración que nos permita hacer pruebas de inicio a fin del nuestro código y así evitar llevar a introducir un error en el app.

Analizando tanto Trunk-Based Development como GitFlow, nos damos cuenta que la diferencia entre estos se basa en el ámbito de desarrollo de una nueva funcionalidad. En el primero se espera que se entreguen porciones de código pequeñas y sean

integradas rápidamente, lo que nos permite ser más efectivos y versátiles al momento de hacer releases, pero a su vez requiere que seamos responsable y que exista automatización que certifique que nuestro código está en el estado óptimo para ser liberado, mientras que GitFlow, si bien puede ser que en general esta metodología puede esperar a que cada desarrollador demore días para entregar la tarea completa, creemos que la complejidad de la misma no tendrá estos tiempos de espera y que al ser pocos integrantes no tendremos grandes de conflictos al mergear a develop. Es por esto que concluimos que utilizar la metodología de GitFlow es lo más acertado en nuestro caso.

Elementos que lo componen

Definimos dos ramas principales por las cuales cada nuevo requerimiento va a pasar antes de salir a producción.

Estas dos ramas van a ser Develop y Main (Ramas de función).

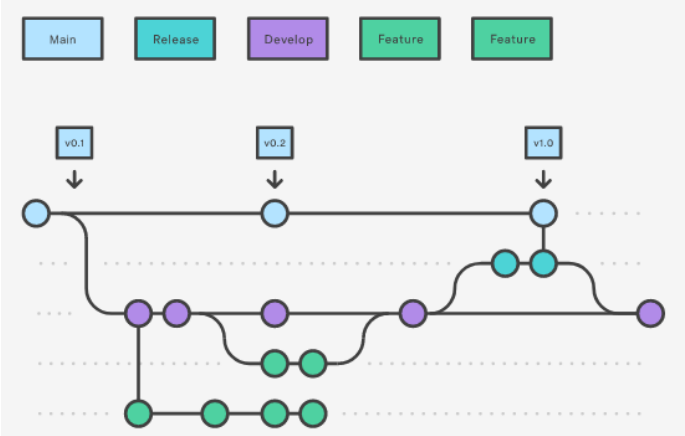
Develop: Esta rama va a contener el proceso continuo de nuestro proyecto. El funcionamiento será que, al ingresar una nueva feature, se crea una nueva rama partiendo desde develop la cual podrá estar abierta a lo largo de un dia, para terminar realizando un merge a Develop, y de esta forma crear una pull request desde Develop hacia Main.

Main: Esta rama es la que contiene la aplicación que está en producción, por lo tanto que ven los usuarios finales y con la cual hay que ser muy cuidadoso. Esta misma va a recibir las pull requests hechas desde Develop.

Ramas de publicación:

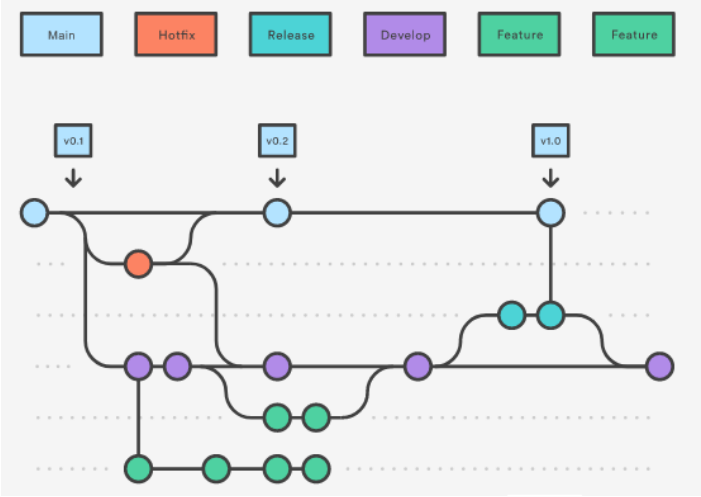
Cuando la rama develop se vaya a publicar, crearemos una rama de publicación (release) a partir de esta. Una vez creada iniciamos el ciclo de publicación, por lo que no podremos agregar nuevas funciones debido a que en esta deben figurar soluciones de errores u otras tareas relacionadas a esta. Cuando esté finalizada, se realiza un merge de la rama release al main etiquetando correctamente y a la vez de fusionar develop nuevamente debido a que esta rama podría haber agregado actualizaciones cuando se inicia el release y las nuevas funcionalidades tiene que poder acceder a estas..

Notar que al igual que las ramas feature, las ramas release se basan en la rama develop.



Ramas de corrección:

Las ramas de corrección (hotfix) sirven para reparar rápidamente las publicaciones de producción. Las ramas hotfix son muy similares a las ramas release y feature, salvo por el hecho de que se basan en la rama main y no en la develop. Esta es la única rama que debería bifurcarse directamente a partir de main. Cuando se haya terminado de aplicar la corrección, debería fusionarse en main y develop (o la rama release actual), y main debería etiquetarse con un número de versión actualizado.



El flujo que vamos a utilizar con la metodología Gitflow es el siguiente:

1. Se crea una rama develop a partir de main.
2. Se crea una rama release a partir de la develop.
3. Se crean ramas feature a partir de la develop.

4. Cuando se termina una rama feature, se fusiona en la rama develop.
5. Cuando la rama release está lista, se fusiona en las ramas develop y main.
6. Si se detecta un problema en main, se crea una rama hotfix a partir de main.
7. Una vez terminada la rama hotfix, esta se fusiona tanto en develop como en main.

Este es básicamente el flujo de trabajo que se sigue cuando se utiliza git flow, es muy ordenado y nos permite llevar un mejor seguimiento del trabajo hecho, y una línea de tiempo más entendible evitando en lo posible los conflictos a la hora de combinar el trabajo hecho.

Análisis de deuda técnica

Testing Exploratorio (Experiencia e interfaz de usuario)

- Cuando se abre por primera vez la pantalla de importadores extensibles, el botón aparece habilitado antes que elijas un importador o “Ninguno”.
- En el login si apretas enter se envían los datos y además se pone en modo visible la contraseña. Si le erraste en tus datos, queda tu contraseña visible.
- Muestra mensajes de error “Server with problems” cuando no es dicho problema, por lo tanto no es útil.
- Al momento de asignar un desarrollador a un proyecto, en el campo para ingresar un desarrollador, no muestra una lista desplegable de los desarrolladores posibles.
- Cuando el usuario ingresa a la sección de Bug y todavía no tiene un proyecto asignado, el error se muestra al final de la pantalla, por lo que el usuario debería scrollear para darse cuenta de que existe un cartel de error y por que es que le aparece ese error.
- Cuando se entra a una sección que contiene una tabla y esta no tiene ningún elemento, se muestran solamente los header. Debería mostrarse un cartel que indique al usuario que el sistema no tiene registros de esos datos.
- En la sección de importadores de bugs, el ejemplo que se ofrece no es correcto ya que la primera sección del path está hardcodeado y en el ejemplo no.
- Si agrego un tester a un proyecto, no me permite volver para atrás, debería salir de la nueva pantalla y buscar el proyecto de nuevo. Debería existir, por ejemplo, un breadcrumb que oriente al usuario
- En la sección de importadores de bugs, el error generado al clickear el hipervínculo “Bug Import Custom”, es incorrecto. Dice que no puede encontrar parte de determinado path, brindándole de esta manera información incorrecta al usuario.

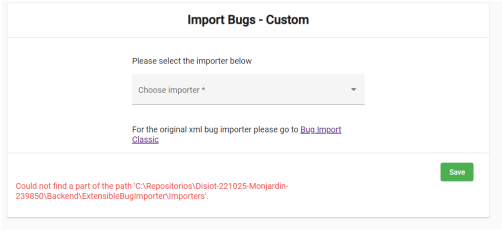
Testing Backend

- Se utilizan ids para los usuarios, pero no se agregó que el nombre de usuario sea único.
- En cada clase del dominio hay un método de validación para cada una de ellas respectivamente, lo cual no está del todo bien ya que las validaciones no tendrían que estar en las clases del dominio sino que en la lógica.

- En varias clases de test, se crean datos de prueba repetidamente para las diferentes funciones en vez de crearse algún objeto global y utilizar el mismo en todas las que se necesite.
- Si un developer resuelve un bug de determinado proyecto y ese proyecto se borra, también se borra el historial de bugs resueltos por el tester
- Si un developer resuelve un bug de determinado proyecto y ese proyecto se borra, también se borra el bug de la base de datos
- Si un developer resuelve un bug de determinado proyecto y ese proyecto cambia de nombre, también se borra el bug de la base de datos
- Si un developer resuelve un bug de determinado proyecto y ese proyecto cambia de nombre,también se borra el historial de bugs resueltos por el tester

Testing Frontend

- Al estar en la sección “Create new user” solamente deja crear un solo Administrador.
- Al crear un bug, poniéndolo como resuelto pero no asignándole ningún desarrollador que lo haya resuelto deja crearlo cuando no debería.
- Al asignar un desarrollador a un proyecto, si se ingresa el nombre de un desarrollador que no exista, lo deja crear igual, lo que no debería.
- En la sección de importadores de bugs, si no hay un importador elegido y se hace click en el botón “save”, queda cargando y no se puede volver a utilizar dicha funcionalidad sin volver hacia atrás.
- En la sección de importadores de bugs, no funciona el hipervínculo de “Bug Import Custom”



- En la sección de importadores de bugs, si el botón “save” está desactivado y se clickea el hipervínculo “Bug Import Custom”, él botón “save” se activa.
- No aparece un mensaje de error al intentar crear un usuario repetido. En cambio, aparece un mensaje de éxito.

Issues (descripción y clasificación)

Para las issues decidimos tener un mismo formato en todas las encontradas para seguir una misma línea de trabajo y que el mismo sea entendible y ordenado.

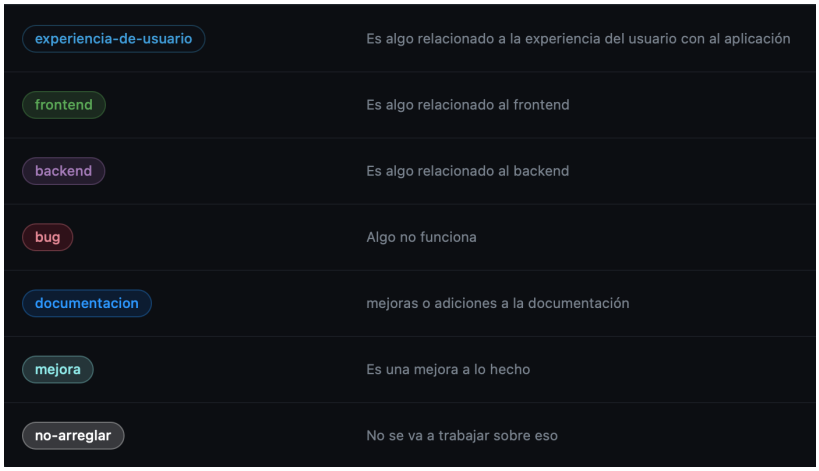
Para cada una de ellas, especificamos su clasificación y descripción. Para aquellas que su clasificación es “bug”, le agregamos una variable severidad.

Nombre

- *Clasificación:*
- *Descripción:*
- *Severidad: (en caso de que sea bug)*

Las etiquetas usadas para ingresar las issues a github fueron:

- bug
- backend
- mejora
- experiencia-de-usuario
- frontend
- documentacion



Resumen de Issues

- **Contraseña visible:**
  - *Clasificación:* Bug
  - *Descripción:* En el login si apretas enter se envían los datos y además se pone en modo visible la contraseña. Si le erraste en tus datos, queda tu contraseña visible.
  - *Severidad:* Baja

- **Botón “save” habilitado en momento incorrecto:**
  - *Clasificación:* Usabilidad
  - *Descripción:* Cuando abrir por primera vez la pantalla de importadores extensibles, el botón aparece habilitado antes que elijas un importador o “Ninguno”.

- **Mensaje de error “Server with problems” incorrecto:**
  - *Clasificación:* Usabilidad
  - *Descripción:* Muestra mensajes de error “Server with problems” cuando no es dicho problema, por lo tanto no es útil.

- **Lista desplegable desarrolladores sin mostrar:**
  - *Clasificación:* Usabilidad
  - *Descripción:* Al momento de asignar un desarrollador a un proyecto, en el campo para ingresar un desarrollador, no muestra una lista desplegable de los desarrolladores posibles.

- **Nombre único:**
  - *Clasificación:* Problemas de estándar.
  - *Descripción:* Se utilizan ids para los usuarios, pero no se agregó que el nombre de usuario sea único.

- **Validaciones en el dominio:**
  - *Clasificación:* Problemas de estándar.
  - *Descripción:* En cada clase del dominio hay un método de validación para cada una de ellas respectivamente, lo cual no está del todo bien ya que las validaciones no tendrían que estar en las clases del dominio sino que en la lógica.

- **Tests repetidos:**
    - *Clasificación:* Problemas de estándar.
    - *Descripción:* En varias clases de test, se crean datos de prueba repetidamente para las diferentes funciones en vez de crearse algún objeto global y utilizar el mismo en todas las que se necesite.

- **Único Administrador:**
    - *Clasificación:* Bug.
    - *Descripción:* Al estar en la sección “Create new user” solamente deja crear un solo Administrador.
    - *Severidad:* Alta.

- **Desarrollador no existente a proyecto:**
    - *Clasificación:* Bug.
    - *Descripción:* Al asignar un desarrollador a un proyecto, si se ingresa el nombre de un desarrollador que no exista, lo deja crear igual, lo que no debería.
    - *Severidad:* Alta.

- **Importación de custom bugs:**
    - *Clasificación:* Bug.
    - *Descripción:* En la sección de importadores de bugs, si no hay un importador elegido y se hace click en el botón “save”, queda cargando y no se puede volver a utilizar dicha funcionalidad sin volver hacia atrás.
    - *Severidad:* Media.

- **Asignación de valores a parámetros de funciones**:
    - *Clasificación*: Problemas de estándar.
    - *Descripció*n: En la lógica de negocio, la clase BugBusinessLogic tiene funciones donde se realiza una asignación dentro de los parámetros de la función, lo cual no cumpliría con la prolijidad del código

- **Metodo muy largo:**
    - *Clasificación*: Problemas de estándar.
    - *Descripció*n: La clase LoginBusinessLogic tiene una función(Login) la cual contiene 20 lineas.La misma no cumple con los estandares de clean code ya que el mismo método tiene más de un nivel de abstracción y, por lo tanto, realiza más de una actividad

- **Mal posicionamiento de error:**
    - *Clasificación*: Problema de comunicación
    - *Descripció*n: Cuando el usuario ingresa a la sección de Bug y todavía no tiene un proyecto asignado, el error se muestra al final de la pantalla, por lo que el usuario debería scrollear para darse cuenta de que existe un cartel de error y por que es que le aparece ese error.

- **Falta texto de error que indique al usuario qué está pasando:**
    - *Clasificación*: Problema de comunicación
    - *Descripció*n: Cuando se entra a una sección que contiene una tabla y esta no tiene ningún elemento, se muestran solamente los header. Debería mostrarse un cartel que indique al usuario que el sistema no tiene registros de esos datos.

- **Mal ejemplo en placeholder:**
    - *Clasificación*: Problema de comunicación
    - *Descripció*n: En la sección de importadores de bugs, el ejemplo que se ofrece no es correcto ya que la primera sección del path está hardcodeado y en el ejemplo no.

- **Mal comunicación de navegación con el usuario:**
    - *Clasificación*: Problema de comunicación
    - *Descripció*n: Si agrego un tester a un proyecto, no me permite volver para atrás, debería salir de la nueva pantalla y buscar el proyecto de nuevo. Debería existir, por ejemplo, un breadcrumb que oriente al usuario

- **Mensaje incorrecto de error. No ayuda al cliente a entender el problema:**
    - *Clasificación*: Problema de comunicación
    - *Descripció*n: En la sección de importadores de bugs, el error generado al clickear el hipervínculo “Bug Import Custom”, es incorrecto. Dice que no puede encontrar parte de determinado path, brindándole de esta manera información incorrecta al usuario.

- **La plataforma deja crear usuarios repetidos:**
    - *Clasificación*: Bug de negocio
    - *Descripció*n: El sistema deja crear usuarios repetidos
    - *Severidad*: Alta.

- **Al borrar un proyecto, se borran los bugs fix de este en el historial del dev que los fixeó:**
    - *Clasificación*: Bug de negocio
    - *Descripció*n: Si un developer resuelve un bug de determinado proyecto y ese proyecto se borra, también se borra el historial de bugs resueltos por el tester
    - *Severidad*: Alta.

- **Al borrar un proyecto, se borran los bugs fix de la base de datos, tabla bugs:**
    - *Clasificación*: Bug de negocio
    - *Descripció*n: Si un developer resuelve un bug de determinado proyecto y ese proyecto se borra, también se borra el bug de la base de datos
    - *Severidad*: Alta.

- **Si se cambia de nombre de un proyecto, se borran los bugs fix de la base de datos, tabla bugs:**
    - *Clasificación*: Bug de negocio
    - *Descripció*n: Si un developer resuelve un bug de determinado proyecto y ese proyecto cambia de nombre, también se borra el bug de la base de datos
    - *Severidad*: Alta.

- **Si se cambia de nombre de un proyecto, se borran los bugs fix de este en el historial del dev que los fixeó:**
    - *Clasificación*: Bug de negocio
    - *Descripció*n: Si un developer resuelve un bug de determinado proyecto y ese proyecto cambia de nombre, también se borra el historial de bugs resueltos por el tester
    - *Severidad*: Alta.

- **Hipervínculo no funciona:**
    - *Clasificación*: Bug de ui
    - *Descripció*n: En la sección de importadores de bugs, no funciona el hipervínculo de “Bug Import Custom”
    - *Severidad*: Media.

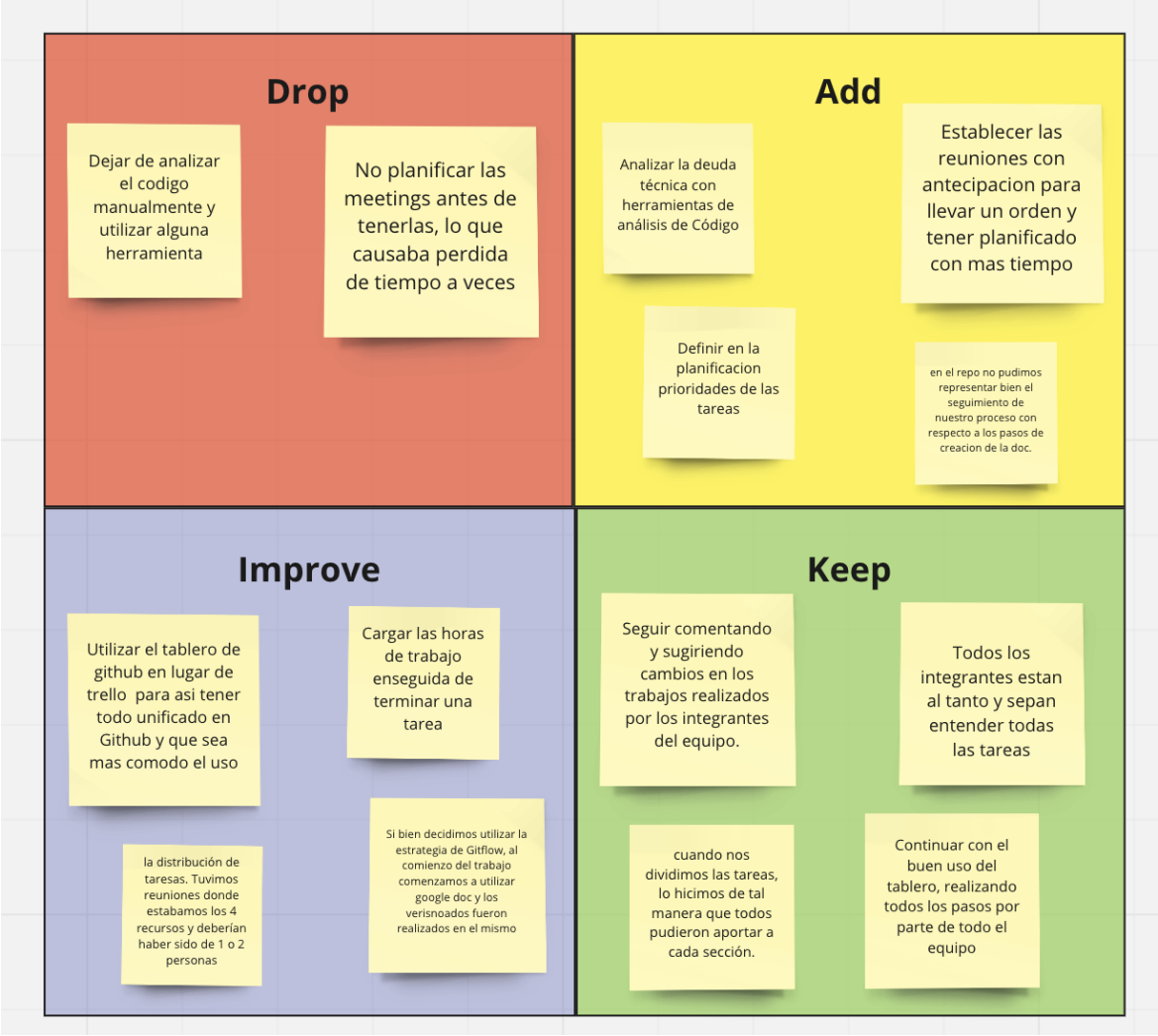
- **Comportamiento incorrecto en la activación y desactivación del botón "save" en “Bug Import Custom”:**
    - *Clasificación*: Bug de ui
    - *Descripció*n: En la sección de importadores de bugs, si el botón “save” está desactivado y se clickea el hipervínculo “Bug Import Custom”, él botón “save” se activa.
    - *Severidad*: Media.

- **Mensaje de éxito cuando debería aparecer un mensaje de error:**
    - *Clasificación*: Bug de ui
    - *Descripció*n: No aparece un mensaje de error al intentar crear un usuario repetido. En cambio, aparece un mensaje de éxito.
    - *Severidad*: Alta.

Retrospectiva basada en DAKI

Link a la grabación de la retrospectiva:
https://www.youtube.com/watch?v=ZioQTc9Dp9Q

Resultado final de la retrospectiva de la tarea 1:



Resultados obtenidos en el periodo

La visión que nos dio la retrospectiva fue qué cosas nos gustaron del proceso y que deberíamos seguir haciendo. Una de ellas fue la forma de trabajo, la cual consiste en no dejar que 1 integrante realizara el trabajo en solitario, sino que todos revisarán lo que fue hecho para poder aportar. Esto permite detectar si algo fue mal confeccionado, está incompleto o incluso mejorar la redacción, aportando un punto de vista diferente.

Dificultades encontradas y formas de solución

Uno de los grandes resultados de esta retrospectiva es la conclusión de no realizar manualmente las mejoras en el código, sino que por comodidad y eficiencia vamos a utilizar alguna herramienta para realizar esta actividad. La falta de uso de herramientas como Ndepend, se debió a que en semestres anteriores se utilizó la misma y no pudimos volver a utilizarla este año por falta de licencia. La PC que se podía usar no tenía los recursos necesarios para su correcta utilización. Para el futuro se actualizará dicha PC (formatear) y agrega hardware acorde para su instalación.
Detectamos una dificultad que fue la mala utilización de los recursos en algunas reuniones donde no se dividió el trabajo correctamente ya que teníamos alguna persona más en una tarea, cuando en realidad ese mismo integrante podía estar avanzando en otra tarea y asi tener mas eficiencia en el flujo de trabajo

Lecciones aprendidas y mejoras en el proceso

Gracias a la retrospectiva, pudimos ver las diferentes visiones de un mismo problema, por lo tanto, se toma conocimiento de las cosas correctas que se hizo en el proceso y mantenerlas, así como también observar las posibles mejoras a realizar y que cosas no se deben hacer. Aprendimos que al tomar una decisión y documentarla, debemos realizar nuestras actividades en base a lo establecido. Como se puede ver, la forma en la que se debería versionar (mediante la utilización de Gitflow) no fue realizada correctamente. También tenemos que anotar en nuestra tabla de excel el tiempo dedicado, enseguida de finalizar una tarea y así evitar estimaciones para poder ser más precisos.
Para concluir, entendemos que el trabajo en esta primera entrega fue correcto ya que cumplimos con todas las pautas solicitadas y no se detectaron problemas que imposibilitaron el accionar del equipo de trabajo, el cual creció y aprendió a aplicar las prácticas de trabajo aprendidas en clase.

Detalle de registro de esfuerzo

Se utilizó una tabla para llevar el registro del tiempo dedicado a esta instancia del obligatorio. La misma lleva el tiempo en minutos por cada integrante.
Cabe destacar que se trabajó en conjunto en varias etapas del mismo.

Tiempo en Minutos				
Día	Rodrigo	Martina	Gabriel	Marcos
jueves 22/09	120	120	120	120
viernes 23/09	60		120	
sabado 24/09				
domingo 25/09	180	180	180	180
lunes 26/09	210	90	150	75
martes 27/09	150	90	90	
miércoles 28/09	60	60		
jueves 29/09				45
viernes 30/09	75	75	75	75
sábado 01/10				60
domingo 02/10	45		30	
Total	900	615	765	555