Javier Esteban Camargo 201313450

David Felipe Sabogal 201327594

Jose Alejandro Silva 201312889

David Camilo Bonilla 201313347

**Entrega Parcial Experimento 3**

Para la entrega parcial del experimento 3 se solicitó evaluar dos requerimientos no funcionales en la aplicación TBC:

* Disponibilidad
* Modificabilidad

1. Disponibilidad

Para la disponibilidad se pide garantizar que ante la falla del servidor central, otra instancia pueda continuar recibiendo la información de los vehículos (posición, alarma, etc.). Se espera que el 99.95% de las tramas de datos enviadas sean atendidas por el servidor.

Para esto se realizó una división en 3 instancias disponibles que se encargan de recibir los datos en locaciones diferentes. La primera es el servidor central. Sincronizados a este y en tiempo real, se generan backups en dos máquinas más, estos servidores recesivos son los encargados de recibir los datos y guardarlos y en caso de falla del servidor central alguno de los dos se vuelve activo, comenzando a recibir todos los datos que no llegan al servidor central.

De esta manera siempre es posible garantizar que la información de los datos llegue a las bases de datos de TBC. En caso de uno de los dos servidores recesivos fallar, el otro servidor recesivo debe volverse activo para recibir los datos que no llegan al servidor central o al primer recesivo. Esto con el fin de dar tiempo suficiente de restablecer el servicio del servidor central.

Generalmente cuando sucede una falla en el servidor central y en alguno de los dos servidores recesivos, los tiempo de respuesta son un poco mayores de lo acostumbrado, más siempre se garantiza que las solicitudes y envío de datos van a tener respuesta.

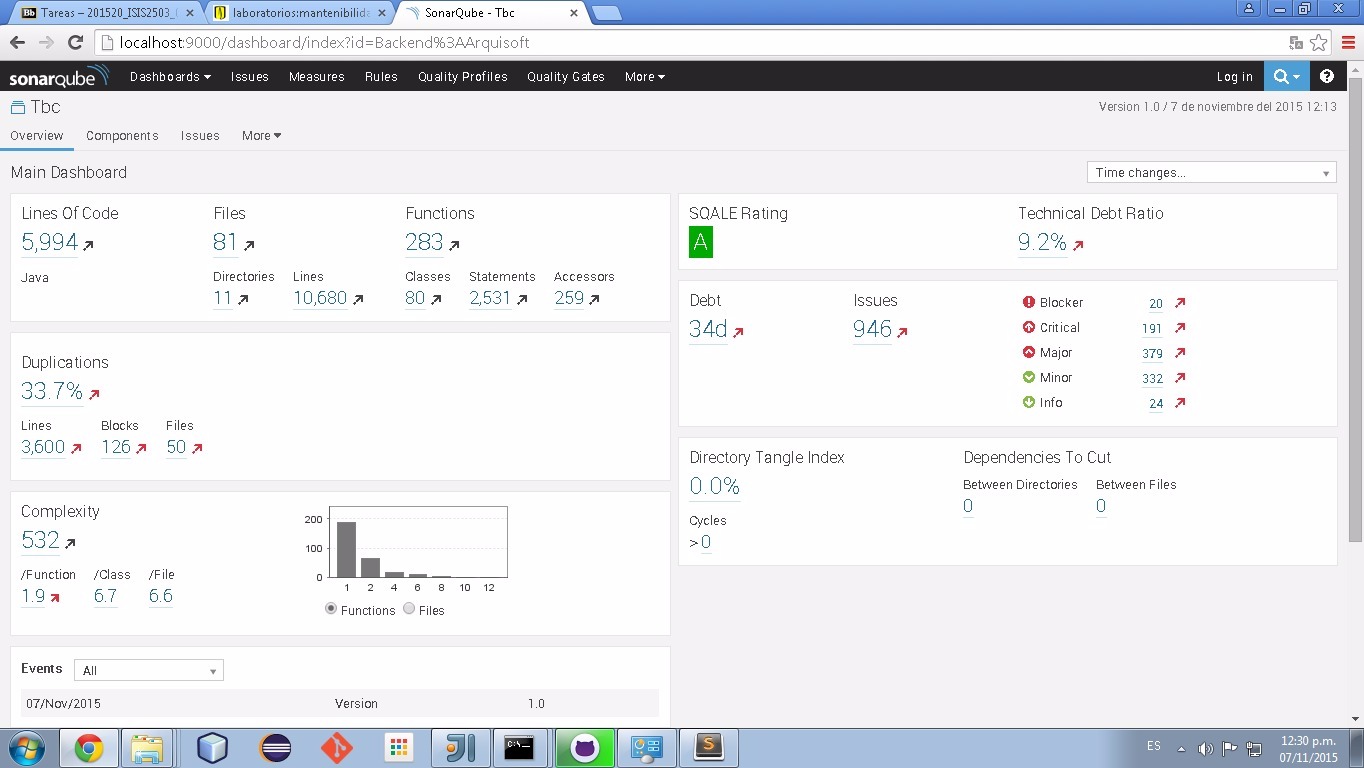
2. Modificabilidad

Se solicitó además el requerimiento no funcional de la modificabilidad. El objetivo es determinar y mejorar la mantenibilidad del desarrollo logrado con el proyecto TBC utilizando una herramienta para analizar y mejorar el código fuente tanto de TBC como de los 3 distintos Standalones, esta es SonarQube.

Para el código fuente del proyecto, se estableció con Sonar las distintas métricas listadas a continuación:

* Complejidad
* Duplicaciones
* Líneas de código
* Issues.

Se procedió a analizar los resultados dados por Sonar y a corregir los errores dados por este.



Como se puede apreciar en la imágen, los errores dados por Sonar eran varios, y graves en su mayoría. Se procedió a corregir tales errores para garantizar la calidad y la modificabilidad del código.

Como prioridad se tomaron los errores catalogados como “blockers” dado a que eran los de mayor alto riesgo para el código, estos eran en su totalidad errores en el envío de excepciones, puesto a que en lugar de estas se estaban retornando ‘Throwables’. El problema se basaba en que los throwables devolvían las subclases “Exception” y “Error” mientras que para el objetivo de nuestra aplicación solo era necesario devolver excepciones.

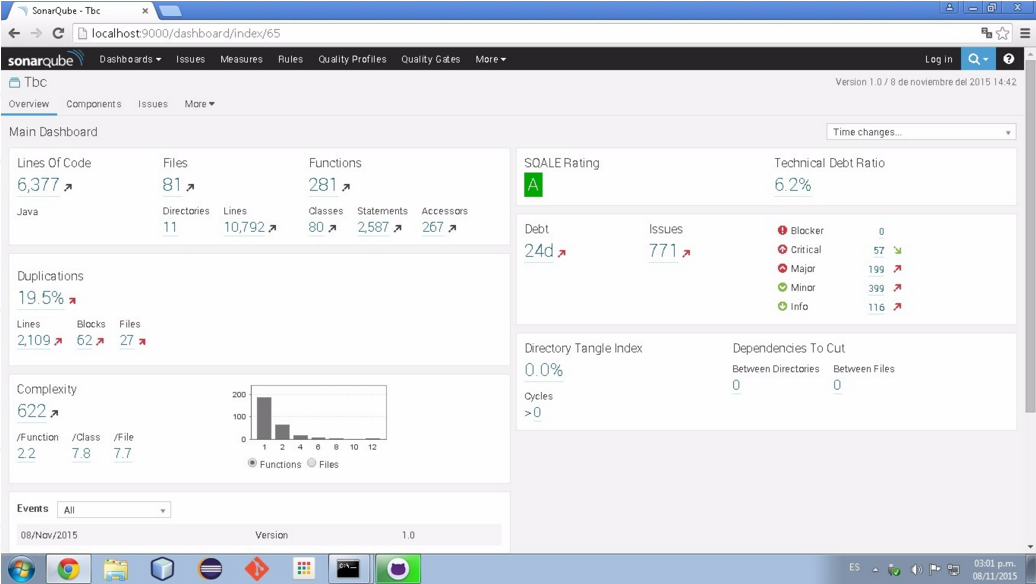
Se procedió a resolver los errores de tipo “Critical” estos eran básicamente errores en las declaraciones en los archivos generados automáticamente. Por ejemplo, se declaraban atributos de tipo “Public static volatile” lo cual es un estilo de contradicción, dado que “public static” implica que el atributo va a ser una constante y “volatile” es una declaración que permite editar. La solución fué declarar los atributos como “public static final”.

Finalmente se comenzó la reducción de errores de tipo “Major” en estos que eran la mayoría se apreciaban duplicaciones, complejidades altas, retornos erróneos en los métodos etc.

Algunos de estos no se pudieron solucionar dado a que algunas de las complejidades altas son necesarias para el correcto funcionamiento de los algoritmos.

Otro ejemplo de errores que no se pudieron solucionar son los de IPs, es decir, Sonar sugería que las IPs enviadas en los métodos fuesen configurables, más en algunos casos es necesario enviar directamente los datos a una IP específica.

Luego de la primera serie de correcciones se obtuvieron los siguientes resultados:



Se puede ver claramente que el proceso de reducción de errores ha sido efectivo, y que de esta manera se garantiza la modificabilidad del código.

Para lograr un código más mantenible, el grupo acordó lograr el menor acoplamiento posible entre los diversos componentes de TBC, enfocados a facilitar el trabajo en equipo y optimizar los procesos de modificación de las diversas clases. Se observa adicionalmente que el nivel de cohesión y de relación entre las clases es bajo, solo se relacionan las clases que estrictamente dependen o necesitan información. Para facilitar la modificabilidad entre clases estrictamente asociadas de TBC se implementó la táctica de utilizar un intermediario, por ejemplo para los Vcub y la organización, la complejidad de dicha asociación era elevada entonces para facilitar la modificabilidad de creo la clase de estación de vcub la cual era un intermediario entre la organización y la clase de Vcub. Adicionalmente se utilizó la técnica de refactorización para facilitar el proceso de modificación de las aplicaciones de StandAlone, se busco crear puntos en común y acompañarlos de manera que seas de fácil conocimiento y modificación. Además dentro de los lineamientos de código se estableció que:

* Todas las excepciones deben ser de tipo Exception más no throwable
* Las declaraciones de atributos se deben hacer de manera adecuada
* No se deben declarar atributos o variables que no se vayan a utilizar puesto a que generan gasto innecesario de memoria
* No se debe duplicar el código para las diferentes clases bajo ninguna circunstancia
* Como máximo se debe restringir el uso de Try, For, If, While a 3 entre todos por método.
* Las declaraciones de atributos de tipo Collection, deben hacerse con clases abstractas y no instancias de estas, por ejemplo “List” en vez de “ArrayList”.
* Para ser más específicos se deben evitar las excepciones genéricas de tipo “Failure, error HTTP code: “