Contenido

[Análisis de métricas: 1](#_Toc500279120)

[Documentación: 1](#_Toc500279121)

[Tamaño: 2](#_Toc500279122)

[Complexity 2](#_Toc500279123)

[Issues 2](#_Toc500279124)

[Duplications 4](#_Toc500279125)

[Tests 4](#_Tests)

[Patrones de diseño utilizados 4](#_Toc500279126)

[Builder: 4](#_Toc500279127)

[Factory 5](#_Toc500279128)

[Wrapper 5](#_Toc500279129)

[Pruebas de Carga 5](#_Toc500279130)

[Sin patrones 5](#_Toc500279131)

[Get 5](#_Toc500279132)

[Post 6](#_Toc500279133)

[Para patrones 7](#_Toc500279134)

[Get 7](#_Toc500279135)

[Post 8](#_Toc500279136)

[Comparación 9](#_Toc500279137)

[Análisis de resultados 10](#_Toc500279138)

# Análisis de métricas:

## Documentación:

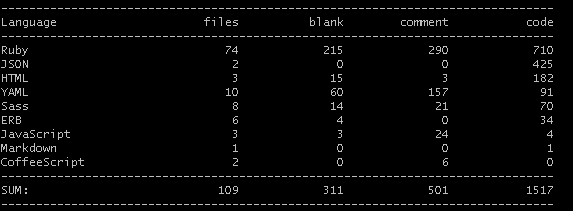
Las métricas relacionadas con la documentación se encargan de expresar la cantidad de espacio en el proyecto dedicado a esta. Aunque en nuestro proyecto no se cuenta con la documentación completa, en un estado ideal de la documentación comparando un antes y un después en cuanto a este aspecto el proyecto, se puede observar que las líneas de código aumentan gracias a la implementación de patrones y por lo tanto la documentación aumenta de la misma manera. Pero, aunque la documentación aumente al aplicar los patrones de diseño, por otro la dificultad en entender la documentación se reduce gracias a que el proyecto gana estructura y las distintas funcionales quedan mejor definidas. Para calculas la densidad de las líneas de documentación se debe seguir la siguiente ecuación:

🡪 Density of comment lines = Comment lines / (Lines of code + Comment lines) \* 100

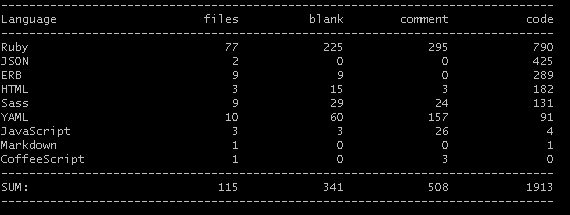
## Tamaño:

El tamaño un proyecto se suele medir por las líneas de código del mismo, haciendo un análisis del antes y el después de aplicar los patrones de diseño, el proyecto creció en 80 líneas de código en Ruby, solo tenemos en cuenta este lenguaje de programación debido a que los tres patrones de diseño que se implementaron, se hicieron en este lenguaje. Respecto a esto, lo que se llevó a cabo fue un Trade-off de tamaño por claridad en la estructura del proyecto. Aunque el proyecto no creció de manera significativa en su tamaño, el mismo logro expresar lo mismo con mayor claridad y por supuesto facilitando la modificabilidad de este para futuros cambios que se le puedan realizar.

Antes de la implementación de los patrones:



Después de la implementación de los patrones:



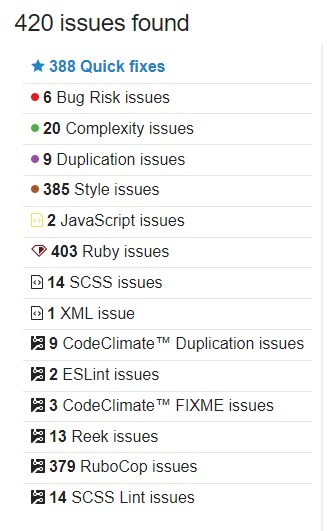
## Complexity

Es una métrica que es en base hacia la complejidad ciclomatica o es conocida como una métrica de McCabe. Esta metrica es medida en el código por sus términos del número de flujos de control que se encuentren. Cada función como mínimo tiene complejidad de uno. Por lo tanto, las palabras claves que se basan en el codigo para calcular su complejidad son los if, for, while, case, catch, throw y return. Sin embargo, palabras como else, default y finally no son contados. Ahora, en el programa se usó varias de estas palabras como el if para identificar usuarios, hacer la autenticación o generar aletas al igual que algunos loops como el while y el for.

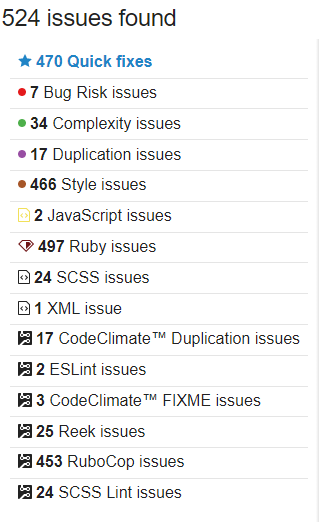
## Issues

La métrica de issues se refiere a las reglas que el código debe cumplir. En este caso se presenta issues en la complejidad, es decir que hay un límite hasta cuanto tiene que tener la complejidad para algunas clases. Hay otros issues que se presenta en ruby y su conexión con la base de datos. Finalmente, también se encuentran issues de la dupplication y bug risks. Para calcular los Issues que tiene el proyecto se utilizó una herramienta llamada ebarapp la cual genero unas pruebas y nos dio el informe de los Issues que contenía el proyecto en total y los que tenía divididos por lenguaje y categoría.

Issues antes de la implementación de los patrones



Issues después de la implementación de los patrones



Al comparar los dos reportes de Issues se puede notar una relación entre la cantidad de Issues que se presentan en el proyecto y el tamaño del proyecto porque a medida que el proyecto se vuelve más robusto y completo el número de reglas que se deben cumplir son mayores.

## Duplications

La duplicación de código se da cuando una parte del código se repite más de una vez dentro del programa, la métrica mide el porcentaje de código repetido en el software. Un alto porcentaje de duplicación de código indica que hay muchas líneas de código innecesarias, lo cual afecta a la modificabilidad del programa ya que si se quieren hacer cambios y esa parte presenta duplicación de código, se va a tener que ir a cada parte y modificarla. Se busca que el código tenga el menor porcentaje de código repetido. Por medio de los patrones de diseño se causa que la duplicación se reduzca ya que el código esta mejor estructurado.

## Tests

Las pruebas en general se realizan para mirar errores o defectos en el software, pero también se tiene que analizar las características de las pruebas. Estas se miden con diferentes métricas como la cobertura, que es la que indica que tantos casos de prueba se van cumpliendo de los especificados, esto se calcula con donde CPE es el número de casos de prueba ejecutados y CPR es el número de casos de prueba que se requieren para cumplir todos los requerimientos. También se tiene la profundidad de las pruebas, el cual muestra que porcentaje de los caminos básicos que se prueban en relación al total de los caminos (Esto es muy ligado a la complejidad ciclomática del programa), esta se calcula donde P es el número de pruebas que se diseñaron y v(g) es la complejidad ciclomática. Además, se puede mirar la madurez de las pruebas, lo cual mide el buen desempeño del flujo de trabajo de las pruebas, ya que también incluye las pruebas que han dado buenos resultados. Esta se calcula donde CPS es el número de casos de prueba positivos y CPR el número de casos de prueba que se requieren para cumplir todos los requerimientos

Patrones de diseño utilizados:

Builder:

Se hizo use del patrón creacional builder para la generación de correos electrónicos que se envían a los supervisores en caso de que se de cualquier tipo de alerta que se le deba notificar. Este patrón consiste básicamente en generalizar algo de lo cual se tienen diferentes instancias y para cada una de estas diferentes instancias se define un proceso de construcción que se base en la generalización del objeto a crear. Lo que se hizo en el proyecto fue generalizar el concepto de correo que se le envía al supervisor al que se le desea comunicar la alerta y para cada distinto tipo de alerta que se puede enviar, se implementó un proceso de construcción particular. Finalmente, lo que se ganó por medio de este patrón fue facilidad en futuras modificaciones que tengan que ver con agregar un nuevo tipo de alerta o mensaje que se le dese notificar a un supervisor, de manera que ahora se puede partir del proceso de construcción que se definió para la generalización del correo que se envía.

## Factory

En este proyecto se decidió usar el patrón de diseño llamado factory. Factory es un patrón creacional, lo cual indica que se divide por jerarquía. Al crear esta jerarquía de objetos se reduce el conjunto de conceptos con los que trabajamos. Por lo tanto, se puede instanciar un objeto a otro dependiendo del tipo. Este patrón se decidió usarlo en las clases de las alertas. Es importante separar las alertas para mantener una mayor organización, en estas y también vale enfatizar que las alertas es lo más importante del programa, ya q tiene como función advertir a los usuarios de posibles riesgos, y de esta manera prevenir causalidades en la mina. El factory tiene una ventaja en estos casos y estos son: ayuda a controlar los objetos que se creen y encapsula la responsabilidad la responsabilidad y el proceso de creación de objetos del producto. En otras palabras, separa los objetos de cada uno y mediante su creación cada objeto separado no afectara el otro. Significando, si en una alerta hay un error, otro tipo de alerta no será afectado por ella. Lo cual esto mejorara el desempeño y la disponibilidad del programa. También el patrón factory hace fácil el intercambio de familias de producto. Es decir, se hace fácil el manejo de información de alertas, mejorando el desempeño. Por último, como es entre cada alerta y no se afectará cada una de ellas si se requiere aumentar microcontroladores o cambiar el formato de las alertas se podrá hacer ya que cada objeto esta por separado.

## Wrapper

Se utilizó el patrón Wrapper en la parte del consumidor para las diferentes peticiones que se deben generar. Generalmente el patrón sirve para adaptar una interfaz cuando una clase que la requiere utilizar no la puede utilizar, entonces convierte una interfaz en otra para que el esta sea la que el cliente espera. En el proyecto se usa una interfaz de mandar request y se usa el Wrapper para adaptar esta interfaz cuando se requiere por HTPP o MQTT. Esto ayuda a la modificabilidad del programa ya que sólo se tiene una interfaz general para las request.

# Pruebas de Carga

## Sin patrones

### Get

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| GET con seguridad | | | | | |
| Etiqueta | # Muestras | Media | Mín | Máx | Desv. Estándar | | % Error |
| registro medicion | 10 | 16 | 10 | 23 | 3.389690251 | | 0 |
| registro medicion | 100 | 15 | 9 | 59 | 7.204685975 | | 0 |
| registro medicion | 500 | 15 | 8 | 75 | 7.789111374 | | 0 |
| registro medicion | 1000 | 15 | 8 | 67 | 8.81697794 | | 0 |
| registro medicion | 5000 | 24 | 7 | 376 | 25.97837439 | | 0 |
| registro medicion | 9000 | 1460 | 993 | 2469 |  | | 3,33% |

### Post

|  |  |  |  |
| --- | --- | --- | --- |
| Post con seguridad | | | |
| Etiqueta | # Muestras | Media | Mín | | Máx | Desv. Estándar | % Error |
| registro medicion | 10 | 26 | 21 | | 46 | 7.203471385 | 0 |
| registro medicion | 100 | 31 | 21 | | 78 | 9.808215944 | 0 |
| registro medicion | 500 | 49 | 32 | | 119 | 14.02455147 | 0 |
| registro medicion | 1000 | 4761 | 48 | | 16929 | 5068.997558 | 0 |
| registro medicion | 5000 | 3935 | 417 | | 26954 | 7470.140441 | 0.852 |

## Para patrones

### Get

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| GET con patrones | | | | | | |
| Etiqueta | # Muestras | Media con seguridad | Mín | Máx | Desv. Estándar | % Error con seguridad |
| registro medicion | 10 | 15 | 10 | 23 | 5.57293053 | 0 |
| registro medicion | 100 | 18 | 9 | 59 | 9.83692046 | 0 |
| registro medicion | 500 | 22 | 8 | 75 | 8.77931492 | 0 |
| registro medicion | 1000 | 23 | 8 | 67 | 8.81697794 | 0 |
| registro medicion | 5000 | 22 | 7 | 376 | 25.97837439 | 0 |
| registro medicion | 10000 | 400 | 200 | 2469 | 67.96478293 | 1% |

### Post

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Post sin seguridad | | | | | | |
| Etiqueta | # Muestras | Media con seguridad | Mín | Máx | Desv. Estándar | % Error con seguridad |
| registro medicion | 10 | 20 | 12 | 46 | 5.2053808 | 0 |
| registro medicion | 100 | 21 | 13 | 155 | 8.916159438 | 0 |
| registro medicion | 500 | 21 | 15 | 200 | 145.0228851 | 0 |
| registro medicion | 1000 | 23 | 20 | 300 | 97658.41241 | 0 |
| registro medicion | 5000 | 24 | 18 | 673 | 44073.79275 | 0 |
| registro medicion | 10000 | 380 | 80 | 1050 | 12867.63022 | 0.09 |

## Comparación

## Análisis de resultados

Al realizar las pruebas del proyecto con los patrones implementados se pueden apreciar dos cambios significativos, ahora las medias de las pruebas son menores gracias a que los patrones como el balanceador de carga ayuda a que el proyecto atienda las solicitudes de manera más eficiente evitando cuellos de botella o sobrecargas sobre algún servidor, además al utilizar el Factory nos facilita la creación y manejo de las alertas lo cual también afecta de manera favorable el desempeño. Complementario a esto los modelos brindan el beneficio de los modelos ahora son mas modificables otro factor que afecta el desempeño