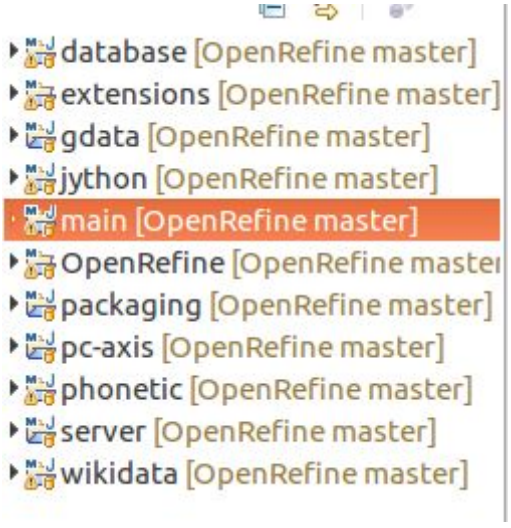
## Sistema Real

### 9. Introdução

O OpenRefine é uma ferramenta baseada em Java que permite clusterizar dados em texto através de diferentes opções, facilitando a interpretação e higienização dos dados. Sendo também muito útil na identificação de anomalias, erros ou casos isolados. É uma ferramenta de código aberto disponível no GitHub , e que atualmente conta com 36 versões lançadas. Os requisitos para utilizar essa ferramenta é ter o JDK8 e Apache Maven instalado no sistema. O OpenRefine é internamente dividido em 11 subprojetos. Nosso foco será o pacote main.

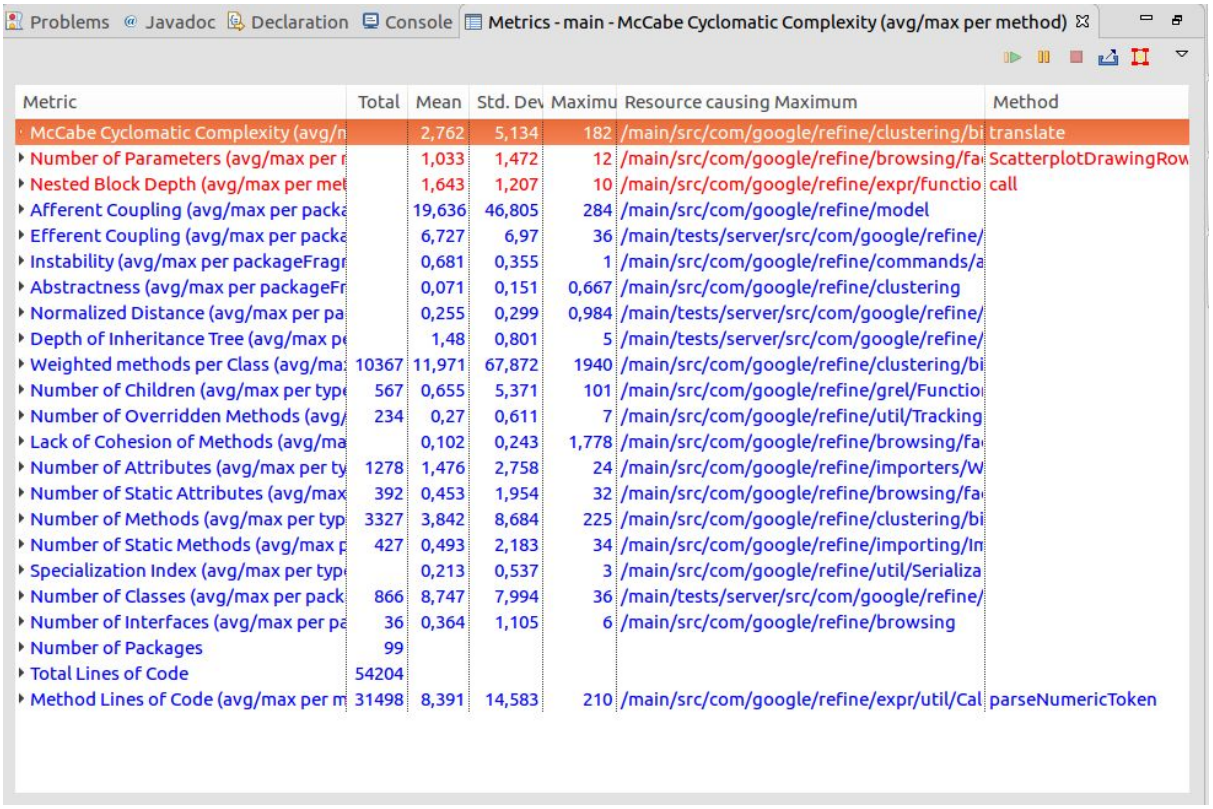



*9.1 Todos os pacotes do projeto OpenRefine*

### 10. Qualidade

#### 10.1. Metrics

O Metrics é uma ferramenta que permite o cálculo de métricas e análise das dependências do projeto. A seguir, vemos as métricas obtidas no pacote main do OpenRefine.




| Metric | Total | Mean | Std. Dev | Maximu | Resource causing Maximum | Method |
|---|---|---|---|---|---|---|
| McCabe Cyclomatic Complexity (avg/n | | 2,762 | 5,134 | 182 | /main/src/com/google/refine/clustering/bi | translate |
| Number of Parameters (avg/max per r | | 1,033 | 1,472 | 12 | /main/src/com/google/refine/browsing/fa | ScatterplotDrawingRow |
| Nested Block Depth (avg/max per met | | 1,643 | 1,207 | 10 | /main/src/com/google/refine/expr/functio | call |
| Afferent Coupling (avg/max per packa | | 19,636 | 46,805 | 284 | /main/src/com/google/refine/model | |
| Efferent Coupling (avg/max per packa | | 6,727 | 6,97 | 36 | /main/tests/server/src/com/google/refine/ | |
| Instability (avg/max per packageFragr | | 0,681 | 0,355 | 1 | /main/src/com/google/refine/commands/a | |
| Abstractness (avg/max per packageFr | | 0,071 | 0,151 | 0,667 | /main/src/com/google/refine/clustering | |
| Normalized Distance (avg/max per pa | | 0,255 | 0,299 | 0,984 | /main/tests/server/src/com/google/refine/ | |
| Depth of Inheritance Tree (avg/max pe | | 1,48 | 0,801 | 5 | /main/tests/server/src/com/google/refine/ | |
| Weighted methods per Class (avg/ma | 10367 | 11,971 | 67,872 | 1940 | /main/src/com/google/refine/clustering/bi | |
| Number of Children (avg/max per type | 567 | 0,655 | 5,371 | 101 | /main/src/com/google/refine/grel/Functio | |
| Number of Overridden Methods (avg/ | 234 | 0,27 | 0,611 | 7 | /main/src/com/google/refine/util/Tracking | |
| Lack of Cohesion of Methods (avg/ma | | 0,102 | 0,243 | 1,778 | /main/src/com/google/refine/browsing/fa | |
| Number of Attributes (avg/max per ty | 1278 | 1,476 | 2,758 | 24 | /main/src/com/google/refine/importers/W | |
| Number of Static Attributes (avg/max | 392 | 0,453 | 1,954 | 32 | /main/src/com/google/refine/browsing/fa | |
| Number of Methods (avg/max per typ | 3327 | 3,842 | 8,684 | 225 | /main/src/com/google/refine/clustering/bi | |
| Number of Static Methods (avg/max p | 427 | 0,493 | 2,183 | 34 | /main/src/com/google/refine/importing/In | |
| Specialization Index (avg/max per type | | 0,213 | 0,537 | 3 | /main/src/com/google/refine/util/Serializa | |
| Number of Classes (avg/max per pack | 866 | 8,747 | 7,994 | 36 | /main/tests/server/src/com/google/refine/ | |
| Number of Interfaces (avg/max per pa | 36 | 0,364 | 1,105 | 6 | /main/src/com/google/refine/browsing | |
| Number of Packages | 99 | | | | | |
| Total Lines of Code | 54204 | | | | | |
| Method Lines of Code (avg/max per m | 31498 | 8,391 | 14,583 | 210 | /main/src/com/google/refine/expr/util/Cal | parseNumericToken |

*10.1 Métricas do pacote principal*

| Nome | Comentário |
|---|---|
| **McCabe Cyclomatic Complexity** | A média no pacote main foi de 2,762, segundo a ferramenta metrics poderia ser menor. |
| **Number of Parameters** | A média foi 1,033. Segundo o metrics a média é alta, mas discordamos, pois apesar de ser um número maior que 1 ainda é um valor baixo, indicando que há pouca dependência no código. |
| **Nested Block Depth** | A média foi de 1,643. O metrics indicou que o resultado está ruim, mas consideramos que, apesar de precisar melhorar, está relativamente bom. |
| **Efferent Coupling** | A média do pacote principal é de 6,727 e está dentro da normalidade. |
| **Afferent Coupling** | A média do pacote principal é de 19,636 e está dentro da normalidade. |

| | |
|---|---|
| **Instability** | A média do pacote principal é de 0,655, ou seja, bem próximo de 0.7 que é considerado ideal. |
| **Abstractness** | A média do pacote principal é de 0,071, bem próximo do extremo 0 que é considerado ideal, visto que a métrica de instabilidade I é próxima de 1. |
| **Normalized Distance** | A média do pacote principal é de 0,255, bem baixa. |
| **Depth of Inheritance** | Temos uma média de 1,48 que é um número baixo, o que implica em menos reutilização de código por herança. |
| **Weighted Methods per Class** | Temos uma média de 11,97 métodos ponderados por classe, o que pode indicar um maior esforço para a manutenção do projeto. |
| **Number of Children** | Temos uma média de 0,65 por classe, ou seja, temos classes com abstrações impróprias das classes pai. |
| **Number of Overridden Methods** | Temos um total de 234 métodos sobrescritos e uma média de 0,27 por classe, um número um tanto mediano e aceitável. |
| **Lack of Cohesion of Methods** | No nosso caso, temos uma média de 0,1 valor LCOM para cada classe do projeto, o que indica que temos classes muito pouco coesas, classes “ruins”. |
| **Number of Attributes** | Aqui temos um total de 1278 atributos e uma média de 1,4 por classe, representando um valor muito baixo dadas as proporções do projeto. |
| **Number of Static Attributes** | Aqui, temos um total de 392 atributos estáticos em todo o projeto com um valor médio de 0,45 por classe. |
| **Number of Methods** | 3,84 métodos em média demonstra classes compactas, tendo em vista que cada método tem em média 8,391 linhas. |

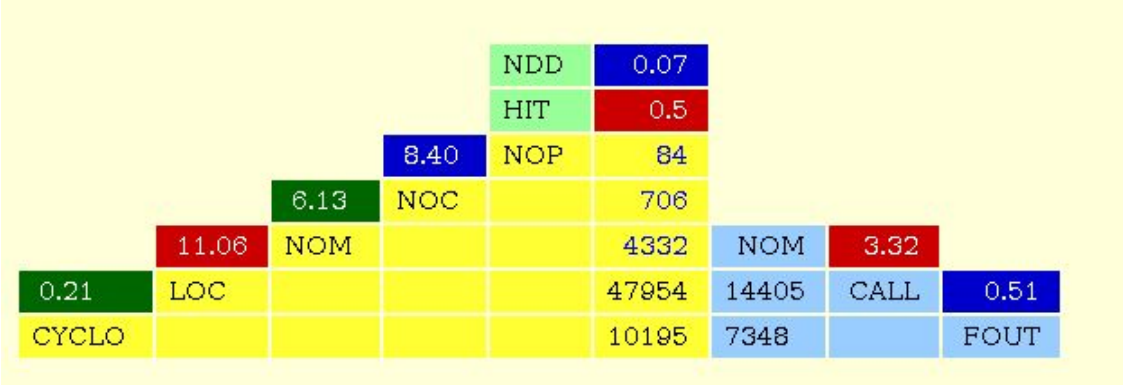
| | |
|---|---|
| **Number of Static Methods** | 0,493 aproximadamente ⅛ dos métodos, de forma que 1 a cada 2 classes em média tem um método estático, o qual dificulta a manutenção. |
| **Specialization Index** | 0,213 passou ser um índice admissível, tendo em vista que as classes possuem poucos métodos. Representando assim poucas substituições de métodos. |
| **Number of Classes** | Com 866 classes, mostra que se trata de um projeto de grandes proporções. |
| **Number of Interfaces** | A presença de 36 interfaces, indica uma possível boa organização do código. Tendo em vista a grande quantidade de classes. |
| **Number of Packages** | A quantidade de pacotes possibilita uma boa organização das 866 classes. Ficando aproximadamente 9 classes por pacote. |
| **Total Lines of Code** | 54204 linhas de código. |
| **Method Lines of Code** | Com uma média de 8,391 linhas por método, demonstra a criação de métodos simples, facilitando a compreensão e manutenção. |

**McCabe Cyclomatic Complexity -** É uma métrica usada para calcular o número de fluxos possíveis de um código fonte.
**Nested Block Depth -** Mede o número de blocos aninhados no código por método.
**Efferent Coupling -** Métrica usada para medir as inter-relações entre as classes.
**Afferent Coupling -** Métrica usada para medir outro tipo de dependências entre pacotes: dependências recebidas.
**Instability -** Métrica indica a uniformidade e organização do código. É a proporção no número de dependências de saída para todas as dependências do pacote.
**Abstractness -** Indica o número de classes abstratas comparado com o número de não abstratas.
**Normalized Distance -** Métrica usada para medir o equilíbrio entre

estabilidade e abstração.
**Depth of Inheritance -** Profundidade de herança é definida como "o comprimento máximo do nó até a raiz da árvore".
**Weighted Methods per Class -** Essa métrica representa a complexidade de uma classe como um todo e pode ser usada para indicar o esforço de desenvolvimento e manutenção da classe.
**Number of Children -** Diz respeito ao número de descendentes imediatos de uma classe e indica o potencial impacto nestes descendentes.
**Number of Overridden Methods -** É o número total de métodos no escopo selecionado que são substituídos por uma classe ancestral.
**Lack of Cohesion of Methods -** Essa é uma medida do número de conjuntos disjuntos formados pela interseção dos conjuntos de variáveis de instâncias usadas pelos métodos.

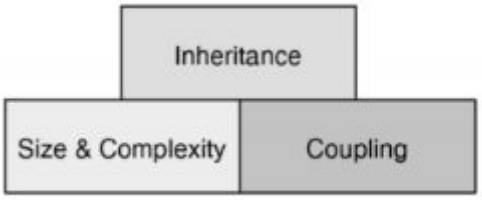
### 10.2. iPlasma

Avaliamos o pacote main usando a ferramenta iplasma que gerou a seguinte pirâmide:

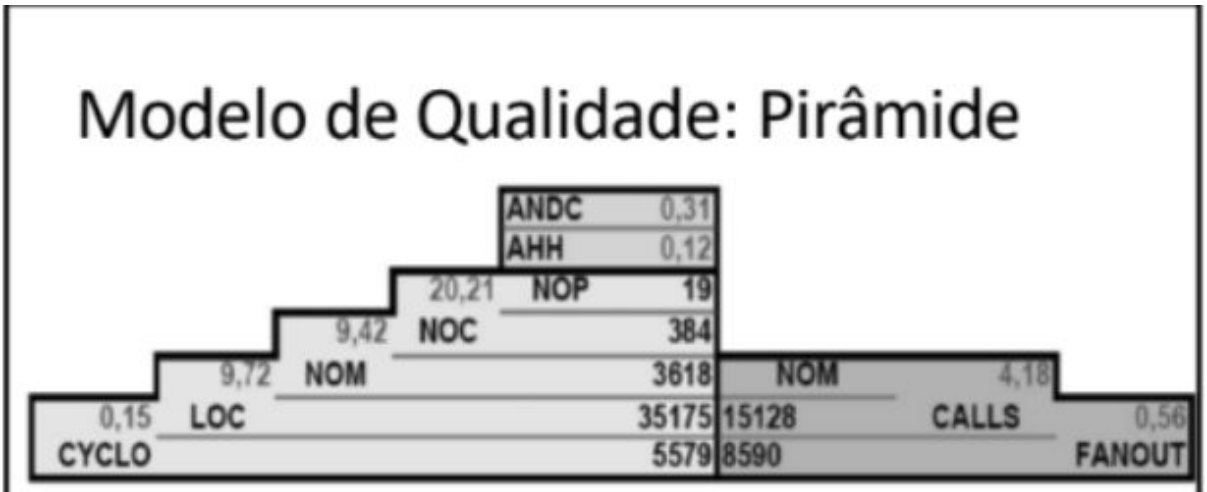



10.1 pirâmide gerada usando a ferramenta iplasma

Ela é dividida em três seções: Herança, Tamanho e complexidade, e Acoplamento. Examinaremos cada uma individualmente.




10.2 seções da pirâmide no iplasma




| | Java | | | C++ | | |
|---|---|---|---|---|---|---|
| Metric | Low | Average | High | Low | Average | High |
| CYCLO/Line of code | 0.16 | 0.20 | 0.24 | 0.20 | 0.25 | 0.30 |
| LOC/Operation | 7 | 10 | 13 | 5 | 10 | 16 |
| NOM/Class | 4 | 7 | 10 | 4 | 9 | 15 |
| NOC /Package | 6 | 17 | 26 | 3 | 19 | 35 |
| CALLS/Operation | 2.01 | 2.62 | 3.2 | 1.17 | 1.58 | 2 |
| FANOUT /Call | 0.56 | 0.62 | 0.68 | 0.20 | 0.34 | 0.48 |
| ANDC | 0.25 | 0.41 | 0.57 | 0.19 | 0.28 | 0.37 |
| AHH | 0.09 | 0.21 | 0.32 | 0.05 | 0.13 | 0.21 |

*10.3 tabela de modelo de qualidade*

### 10.2.1. Herança (NDD e HIT)

Duas métricas são consideradas nessa seção da pirâmide: NDD e HIT. A primeira, NDD, indica o número de descendentes diretos, ou seja, o percentual de subclasses presentes no módulo. O resultado do pacote main foi de 0.07, ou seja 7% do total de classes são subclasses. De acordo com a tabela de referência 1, o valor é um ponto positivo, pois está muito abaixo da média.

A outra métrica, HIT (Altura Hierárquica Média), indica a profundidade média da estrutura hierárquica de herança do projeto. Para o pacote main, o valor obtido foi 0.5 e segundo a tabela é alto demais. Isso indica que a estrutura hierárquica de classes está desequilibrada.

### 10.2.2. Tamanho e Complexidade ( NOP, NOC, NOM, LOC, CYCLO)

Essa seção possui 5 métricas que juntas fornecem informações acerca do tamanho e complexidade dos pacotes, classes e métodos do projeto. São elas:

- **NOP :** Número total de pacotes (84);
- **NOC :** Número total de classes (708);
- **NOM :** Número total de métodos (4332);
- **LOC :** Número total de linhas de código (47954);
- **CYCLO :** Complexidade ciclomática (10195).
- **NOC / NOP:** Indica a média de classes por pacote. O valor obtido foi de 8.40, que segundo a tabela, é um valor razoável, pois não está muito baixo e nem muito alto.
- **NOM / NOC:** Indica a média de métodos por classe. Obtemos o valor de 6.13 e segundo a tabela está dentro da média. Isso significa que provavelmente existe um bom número de métodos por classe.
- **LOC/NOM :** Indica a média de linhas de código por cada método. Obtemos o valor 11.06. Não foi um bom valor, pois segundo a tabela é um valor alto, indicando que os métodos são longos e poderiam ser menores.
- **CYCLO/LOC :** Indica o nível de complexidade médio por cada operação. Obtemos o valor 0.21, que está próximo da média.

### 10.2.3 Acoplamento (CALLS E FOUT)

A última sessão da pirâmide fornece informações sobre o nível de acoplamento das classes do pacote. A métrica CALLS representa o número de chamadas de métodos e operações e a FOUT o número de tipos que são referenciados por classes ou interfaces. A última desconsidera tipos que fazem parte da mesma hierarquia.

- **CALLS / NOM:** Indica a intensidade de acoplamento. Obtemos 3.32 e segundo a tabela é um valor alto.
- **FOUTS / CALLS:** Indica o número médio de classes envolvidas em chamadas de métodos e operações. Obtemos 0.51, número baixo segundo a tabela

## 10.3 Spot Bugs

A ferramenta Spot Bugs divide os tipos de bugs / más práticas em três categorias: Scary, Troubling e Of Concern. Nos quais foram detectados 85 alertas ao total, dos quais destacamos 30 alertas que podem ser conferidos aqui, separados por suas categorias e tipos detalhados a seguir.

### 10.3.1 Scary

No pacote principal (main) foram detectados 21 alertas para essa categoria. Onde destacamos os tipos:

- **BC IMPOSSIBLE INSTANCEOF (Correctness):** instanceof sempre vai retornar false. Isso pode ser uma indicação de algum mal-entendido ou erro de lógica.
- **ICAST INT CAST TO FLOAT PASSED TO ROUND (Correctness):** valor inteiro convertido para ponto flutuante e depois passado para Math.round
- **NP NULL ON SOME PATH (Correctness):** Possível desreferência de ponteiro nulo de valores.
- **NP NULL PARAM DEREF (Correctness) :** Chamada de método que passa um valor nulo para um parâmetro de método não nulo.
- **DMI INVOKING TOSTRING ON ARRAY (Correctness) :** O código chama toString em um array, o que gerará um resultado bastante inútil, como [C@16f0472.
- **MF CLASS MASKS FIELD (Correctness) :** Indica que uma classe define um campo com o mesmo nome que um campo de instância visível em uma superclasse. Isso é confuso e pode indicar um erro se os métodos atualizarem ou acessarem um dos campos quando desejarem o outro.

### 10.3.2 Troubling

No pacote principal (main) foram detectados 12 alertas para essa categoria. Onde destacamos os tipos:

- **ES COMPARING STRINGS WITH EQ (Bad practice):** Comparação de objetos Strings usando o operador == ou !=
- **UWF UNWRITTEN FIELD (Correctness) :** Campo que nunca é gravado. Todas as leituras a determinado campo retornam o valor

padrão. É necessário verificar se há erros (deveria ter sido inicializado?) Ou remover se for inútil.

- **UC USELESS CONDITION (Style) :** Aparece quando uma condição sempre produz o mesmo resultado que o valor da variável envolvida que foi reduzida anteriormente. Provavelmente outra coisa foi criada ou a condição pode ser removida.

### 10.3.3 Of Concern

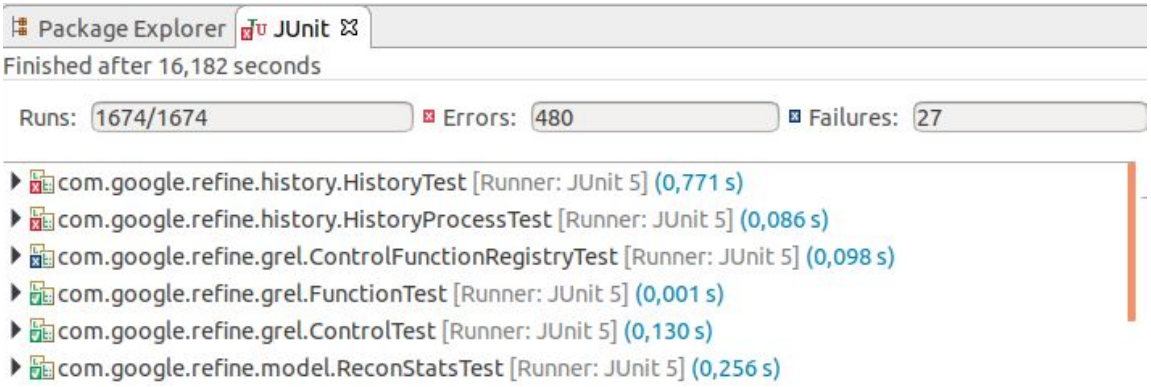
No pacote principal (main) foram detectados 52 alertas para essa categoria. Onde destacamos o tipo:

- **ST WRITE TO STATIC FROM INSTANCE METHOD:** A gravação em um campo estático de um método de instância é propensa a condições de corrida, a menos que você use a sincronização. Além disso, torna difícil manter o estado estático consistente e afeta a legibilidade do código.

## 11. Testes

### 11.1 CodePro

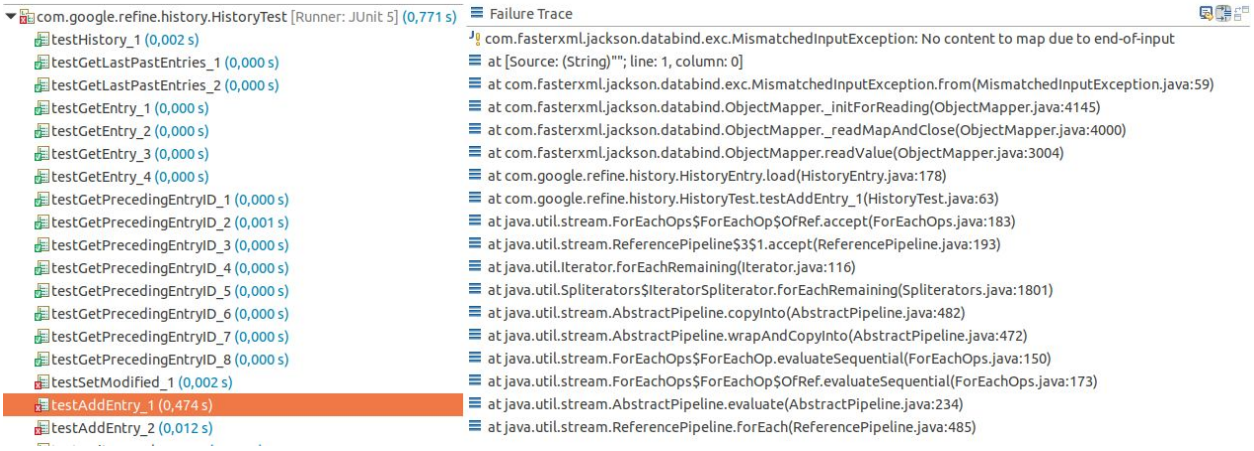
Para gerar testes automáticos do OpenRefine foi utilizada a ferramenta CodePro como plugin do Eclipse para projetos Maven, dentro das restrições da ferramenta o escopo escolhido para geração dos testes foi o subprojeto main. A ferramenta faz uma análise estática do código e então gera os casos de testes (Junit) com base na análise. A configuração da ferramenta para geração de testes foi a padrão, em que foram geradas 137 classes de testes, com um total de 1674 (mil seiscentos e setenta e quatro) testes gerados automaticamente, após a execução que durou aproximadamente 1 hora. Os testes gerados podem ser encontrados aqui. Os resultados apresentados pela execução desses testes foram os seguintes: 480 testes com erros, 27 com falha e 1167 que passaram com sucesso.




*11.1 - Execução dos testes gerados pelo CodePro*

### 11.1.1 Testes com Erros

Os testes com erro não detectaram bugs no sistema. Todavia, lançaram exceções *NullPointerException* ou apresentaram inconsistências de dados, haja vista a utilização de classes e subpacotes dos pacotes master *org.apache.** , *javax.servlet.** , *com.fasterxml** que não conseguiram ser importados no projeto (mesmo manualmente).




*11.2 - Exemplo de Rastreamento de falha de teste com erro*

```
12 import org.apache.commons.collections4.ArrayStack;
13 import org.apache.commons.io.input.BrokenInputStream;
14 import org.apache.commons.io.output.NullWriter;
15 import org.apache.thrift.TByteArrayOutputStream;
16 import org.junit.After;
17 import org.junit.Before;
18 import org.junit.Test;
```

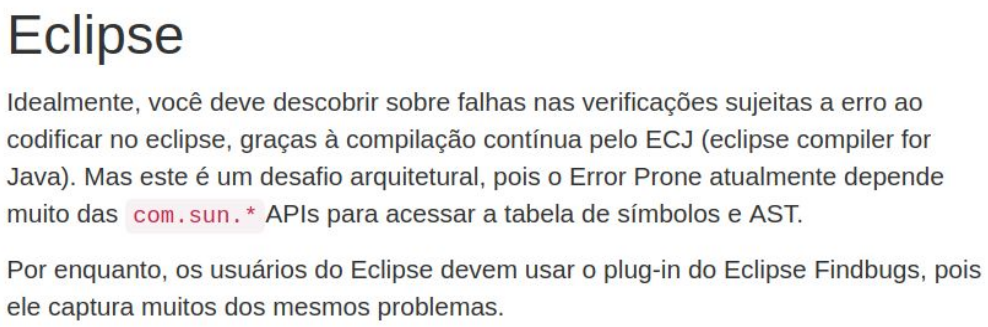
*11.3 - Exemplo de erro de importação do teste acima*

### 11.1.2 Testes com Falhas

Os testes que tiveram resultados diferentes do esperado foram sinalizados como falhos (alguns) em decorrência dos erros de importação descritos acima, o que acabou gerando inconsistência de dados. Outros apresentaram resultados diferentes do que se esperava, o que pode sinalizar bugs no sistema, mas que não haviam sido detectados pelas métricas analisadas na fase de Gerência de Qualidade do Software.

## 11.2 Error Prone

Haja vista a imagem abaixo que se refere à instalação da ferramenta Error Prone (vide site), optamos por seguir o conselho de utilização do Findbugs que acaba capturando, basicamente, os mesmos erros que o Error Prone capturaria. O Findbugs sofreu uma evolução de Software e hoje se chama Spotbugs, o qual foi utilizado na área de Gerência de Qualidade do Software Real (seção 10.3).




*11.4 - Texto retirado do site do Error Prone*

## 11.3 PMD

O PMD é um analisador de código-fonte estático de código aberto que relata problemas encontrados no código do aplicativo. O mesmo ao ser executado divide a análise em 5 tipos de violação que serão listadas aqui de acordo com a execução feita para o OpenRefine.

## 11.4 FBInfer

A FBInfer é uma ferramenta de análise de códigos java. Para efetuar essa análise no projeto OpenRefine é necessário a compilação do projeto através da mesma. Todavia, a ferramenta não conseguiu

efetuar a compilação do projeto e nem foi capaz de fazer a análise do projeto com o mesmo já compilado sem ser pela ferramenta FBInfer.

O projeto OpenRefine utiliza a ferramenta de automação de compilação Maven e o FBInfer oferece suporte para compilação via Maven. Mas, apesar da compilação ocorrer normalmente quando utilizado somente a compilação via Maven, quando se faz a compilação através do FBInfer para projetos com Maven conforme instruído na sua documentação no seguinte link, o mesmo não consegue efetuar a compilação e análise do código.

Entramos em contato com os desenvolvedores do FBInfer sobre esse problema, mas até a conclusão deste relatório ainda não obtivemos resposta.

### 11.5 Coverage

Foi utilizado a ferramenta Coverage presente por padrão na IDE eclipse na versão 2019-09 R (4.13.0) para fazer a análise de cobertura dos testes gerados pela ferramenta CodePro supracitados. A análise obtida pela Coverage foi a descrita na tabela a seguir:

| Element | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
|---|---|---|---|---|
| main | 10,2 % | 11.232 | 99.209 | 110.441 |

Notamos que os 1674 testes gerados pela ferramenta CodePro para o componente main do OpenRefine tiveram uma cobertura de 10,2% de todas as instruções presentes no componente main, o que representa uma cobertura percentual baixa (longe dos 100% ideal), todavia se apresentando muito útil, já que se trata de um teste de 11.232 instruções.

Tentamos a utilização de outras ferramentas de geração automática de testes visando o aumento da cobertura dos testes. Porém as ferramentas Randoop e Evosuite não foram capazes de gerar testes para o componente testado, nos indicando assim uma certa dificuldade para se gerar testes automáticos para esse componente. Isso nos leva a uma explicação plausível para a baixa cobertura dos testes gerados

pela ferramenta CodePro, a única entre as 3 citadas que foi capaz de gerar testes automaticamente.

## 12. Evolução

### 12.1 Evolução usando a ferramenta SpotBugs

#### 12.1.1 Correção do ES_COMPARING_STRINGS_WITH_EQ:

Corrigimos a má prática de comparação de Strings que usa o operador == alterando para o uso do equals. Fizemos a alteração e enviamos uma pull request para o repositório original, a pull request foi aceita.
ver modificações
Como se pode ver pela ultima imagem do anexo, a modificação passou em todos os testes, portanto o comportamento foi preservado.

#### 12.1.2 Correção do INT_BAD_COMPARISON_WITH_NONNEGATIVE_VALUE:

O programa tinha um bug quando o tamanho do array fieldLengths era igual a zero, pois o acesso a fieldLengths[0] era impossível, gerando um IndexOutOfBoundsException. Fizemos a alteração necessária e enviamos uma pull request para o repositório original, essa pull request também foi aceita.
ver modificações
Como se pode ver pela ultima imagem do anexo, a modificação passou em todos os testes, portanto o comportamento foi preservado.

#### 12.1.3 Correção do NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE:

O programa tinha trechos que não verificavam se o retorno de um método era null. Isso possívelmente levaria a uma NullPointerException quando o código fosse executado. Resolvemos esse bad smell fazendo a verificação apropriada e enviamos uma pull request para o repositório original.
ver modificações
Como se pode ver pela ultima imagem do anexo, a maior parte da modificação passou em todos os testes, preservando o comportamento. Uma parte menor do refactoring não passou em um teste, mas acreditamos que o refactoring foi válido (uma explicação para isso se encontra no anexo).

#### 12.1.4 Correção do UC_USELESS_CONDITION:

O programa tinha um trecho com uma condição inútil, observe pela figura que nesse ponto sabemos que m_current != 3 . Resolvemos

esse bad smell removendo a condição desnecessária. Alteramos e enviamos uma pull request para o repositório original.
ver modificações
Como se pode ver pela ultima imagem do anexo, a modificação passou em todos os testes, portanto  o comportamento foi preservado.

### 12.1.5 Correção do RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE:

O programa tinha um trecho de código que checava se um valor era nulo em determinado momento , mas nesse momento era impossível ser nulo por que já foi desreferenciado e se fosse uma Exceção teria acontecido, o que não aconteceu. Resolvemos esse bad smell removendo a verificação desnecessária. Alteramos e enviamos uma pull request para o repositório original.
ver modificações
Como se pode ver pela ultima imagem do anexo, a modificação passou em todos os testes, portanto  o comportamento foi preservado.

## 12.2 Análise da Refatoração

| Metric | Total | Mean | Std. Dev | Maximu |
|---|---|---|---|---|
| McCabe Cyclomatic Complexity (avg/n | | 2,762 | 5,134 | 182 |
| Number of Parameters (avg/max per r | | 1,033 | 1,472 | 12 |
| Nested Block Depth (avg/max per met | | 1,643 | 1,207 | 10 |
| Afferent Coupling (avg/max per packa | | 19,636 | 46,805 | 284 |
| Efferent Coupling (avg/max per packa | | 6,727 | 6,97 | 36 |
| Instability (avg/max per packageFragr | | 0,681 | 0,355 | 1 |
| Abstractness (avg/max per packageFr | | 0,071 | 0,151 | 0,667 |
| Normalized Distance (avg/max per pa | | 0,255 | 0,299 | 0,984 |
| Depth of Inheritance Tree (avg/max pe | | 1,48 | 0,801 | 5 |
| Weighted methods per Class (avg/ma | 10367 | 11,971 | 67,872 | 1940 |
| Number of Children (avg/max per type | 567 | 0,655 | 5,371 | 101 |
| Number of Overridden Methods (avg/ | 234 | 0,27 | 0,611 | 7 |
| Lack of Cohesion of Methods (avg/ma | | 0,102 | 0,243 | 1,778 |
| Number of Attributes (avg/max per ty | 1278 | 1,476 | 2,758 | 24 |
| Number of Static Attributes (avg/max | 392 | 0,453 | 1,954 | 32 |
| Number of Methods (avg/max per typ | 3327 | 3,842 | 8,684 | 225 |
| Number of Static Methods (avg/max p | 427 | 0,493 | 2,183 | 34 |
| Specialization Index (avg/max per type | | 0,213 | 0,537 | 3 |
| Number of Classes (avg/max per pack | 866 | 8,747 | 7,994 | 36 |
| Number of Interfaces (avg/max per pa | 36 | 0,364 | 1,105 | 6 |
| Number of Packages | 99 | | | |
| Total Lines of Code | 54204 | | | |
| Method Lines of Code (avg/max per m | 31498 | 8,391 | 14,583 | 210 |

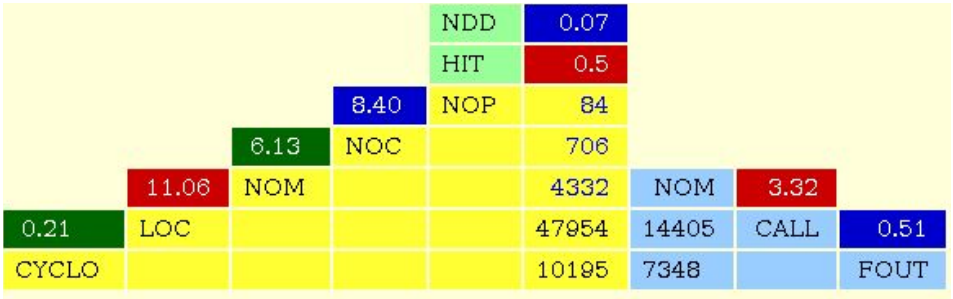
*12.1 Métricas do pacote principal antes da refatoração*

| Metric | Total | Mean | Std. Dev. | Maximum |
|---|---|---|---|---|
| McCabe Cyclomatic Complexity (avg/n | | 2,756 | 5,124 | 182 |
| Number of Parameters (avg/max per r | | 1,031 | 1,469 | 12 |
| Nested Block Depth (avg/max per met | | 1,64 | 1,206 | 10 |
| Afferent Coupling (avg/max per packa | | 21,414 | 51,741 | 330 |
| Efferent Coupling (avg/max per packa | | 6,737 | 6,963 | 36 |
| Instability (avg/max per packageFragr | | 0,672 | 0,358 | 1 |
| Abstractness (avg/max per packageFr | | 0,071 | 0,151 | 0,667 |
| Normalized Distance (avg/max per pa | | 0,264 | 0,301 | 0,985 |
| Depth of Inheritance Tree (avg/max pe | | 1,597 | 0,939 | 5 |
| Weighted methods per Class (avg/ma | 10391 | 11,971 | 67,801 | 1940 |
| Number of Children (avg/max per type | 568 | 0,654 | 5,366 | 101 |
| Number of Overridden Methods (avg/ | 234 | 0,27 | 0,61 | 7 |
| Lack of Cohesion of Methods (avg/ma | | 0,103 | 0,245 | 1,778 |
| Number of Attributes (avg/max per ty | 1284 | 1,479 | 2,758 | 24 |
| Number of Static Attributes (avg/max | 394 | 0,454 | 1,953 | 32 |
| Number of Methods (avg/max per typ | 3343 | 3,851 | 8,678 | 225 |
| Number of Static Methods (avg/max p | 428 | 0,493 | 2,185 | 34 |
| Specialization Index (avg/max per type | | 0,222 | 0,542 | 3 |
| Number of Classes (avg/max per pack | 868 | 8,768 | 7,982 | 36 |
| Number of Interfaces (avg/max per pa | 36 | 0,364 | 1,105 | 6 |
| Number of Packages | 99 | | | |
| Total Lines of Code | 54419 | | | |
| Method Lines of Code (avg/max per m | 31585 | 8,376 | 14,561 | 210 |

*12.2 Métricas do pacote principal depois da refatoração*

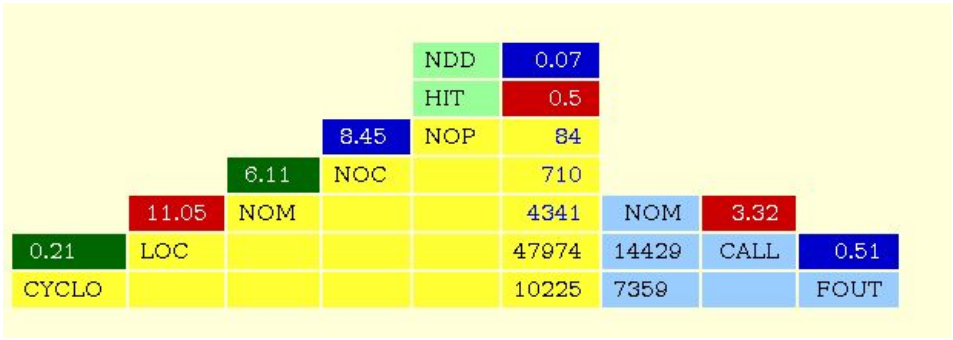
Observando as duas imagens, vemos que a métrica instabilidade melhorou, caiu de 0.681 para 0.672. A métrica de cyclomatic complexity também melhorou um pouco, de 2.762 para 2.756.

### 12.2.1 Antes da refatoração




*12.3 pirâmide antes da refatoração*

### 12.2.2 Depois da refatoração




*12.4 pirâmide depois da refatoração*

A diferença entre as duas pirâmides não foi tão grande. Mas isso se deve ao fato de que o número de linhas e métodos modificados é muito pequeno  quando comparado ao número de linhas e métodos do projeto por completo. Mesmo assim dá para notar que a métrica **LOC/NOM** teve uma leve melhora de 11.06 para 11.05 e que as métricas **NOM/NOC** e **NOC/NOP** mudaram mas continuam dentro da média.