**UFSCAR – Universidade Federal de São Carlos**

**Departamento de Computação**

**Engenharia de Software 2 – 2017/1**

**Relatório Trabalho Final**

Eduardo Noronha RA: 551520

Guilherme Rodrigues RA: 552020

Gustavo Braghim RA:

Lucas Bataglia RA:

Lucas Pessoa RA:

William Piccon RA:

**Introdução**

Esse trabalho tem como objetivo colocar em prática todos as técnicas e conhecimentos aprendidos durante o curso de Engenharia de Software 2 realizado no primeiro semestre de 2017.

O projeto foi dividido em etapas que serão explicadas nos próximos tópicos do trabalho, dentre há uma série de manutenções no software JabRef e realizações de testes para verificar a efetividade dessas manutenções, além de gerar relatórios utilizando a ferramenta Sonarqube.

Para isso, utilizaremos a linguagem Java e a IDE IntelliJ IDEA. Todo o código do projeto assim como as etapas de desenvolvimento podem ser verificadas em (**COLOCAR LINK DO REPOSITORIO**).

**Etapa 1 – Testes Funcionais**

Durante essa etapa foram desenvolvidos testes para verificar os campos durante a inserção de um item bibliográfico(Bibtex) para as categorias book e article, e verificar a importação de arquivos no database atual.

Após a criação será necessário gerar um relatório de cobertura de código gerado pelo software Sonarqube.

* 1. **Testes Funcionais de item bibliográfico na categoria book**

Durante os testes funcionais da categoria book, foram criados testes para verificar se todos os campos são válidos, verificar quando nenhum campo foi preenchido e verificar quando os campos opcionais também foram preenchidos.

O código dos testes pode ser verificado abaixo:

@Test  
public void todosCampos() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{1, title = {São Marcos},\n" + "publisher = {Mauro Beting},\n year = {2012},\n author={Mauro},\n editor = {Abril} }");  
  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setCiteKey("1");  
 entradaValida.setField("title", "São Marcos");  
 entradaValida.setField("publisher", "Mauro Beting");  
 entradaValida.setField("year", "2012");  
 entradaValida.setField("author", "Mauro");  
 entradaValida.setField("editor", "Abril");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
@Test  
public void tNull() throws ParseException{  
 List<BibEntry> test2 = parser.parseEntries("@book{,}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test2);  
}

@Test  
public void camposExtras() throws ParseException{  
 List<BibEntry> test3 = parser.parseEntries("@book{2, title = {Na natureza Selvagem},\n" + "publisher = {Joseph},\n year = {2018},\n author={Michael Johnson},\n editor = {Abril},\n "+ "volume = {1},\n series = {2},\n edition = {6},\n note = {Livro legal}, number = {7},\n"+ "address = {UFSCar},\n month = {Fevereiro}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setCiteKey("2");  
 entradaValida.setField("title", "Na natureza Selvagem");  
 entradaValida.setField("publisher", "Joseph");  
 entradaValida.setField("year", "2018");  
 entradaValida.setField("author", "Michael Johnson");  
 entradaValida.setField("editor", "Abril");  
 entradaValida.setField("volume", "1");  
 entradaValida.setField("series", "2");  
 entradaValida.setField("edition", "6");  
 entradaValida.setField("note", "Livro legal");  
 entradaValida.setField("number", "7");  
 entradaValida.setField("address", "UFSCar");  
 entradaValida.setField("month", "Fevereiro");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test3);  
}

* 1. **Testes Funcionais de um item bibliográfico na categoria article**

Durante os testes funcionais da categoria article, foram criados testes para verificar se todos os campos são válidos, verificar quando nenhum campo foi preenchido e verificar quando os campos opcionais também foram preenchidos.

O código dos testes pode ser verificado abaixo:

@Test  
public void todosCampos() throws ParseException{  
 List<BibEntry> teste1 = parser.parseEntries("@article{1, author={Gustavo},\n"+ "title = {A descoberta dos dados},\n journal = {json},\n year = {2017} }");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("article");  
 entradaValida.setCiteKey("1");  
 entradaValida.setField("author", "Gustavo");  
 entradaValida.setField("title", "A descoberta dos dados");  
 entradaValida.setField("journal", "json");  
 entradaValida.setField("year", "2017");  
 *assertEquals*(Collections.*singletonList*(entradaValida), teste1);  
}  
  
@Test  
public void tNull() throws ParseException{  
 List<BibEntry> teste2 = parser.parseEntries("@article{,}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("article");  
 *assertEquals*(Collections.*singletonList*(entradaValida), teste2);  
}  
  
@Test  
public void camposExtras() throws ParseException{  
 List<BibEntry> teste3 = parser.parseEntries("@article{2, author={Lucas},\n"+ "title = {Engenharia de Software do futuro},\n journal = {json},\n year = {2017},\n"+ "volume = {1},\n number = {1},\n pages = {fifth},\n month = {february},\n note = {lalala} }");  
  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("article");  
 entradaValida.setCiteKey("2");  
 entradaValida.setField("author", "Lucas");  
 entradaValida.setField("title", "Engenharia de Software do futuro");  
 entradaValida.setField("journal", "json");  
 entradaValida.setField("year", "2017");  
 entradaValida.setField("volume", "1");  
 entradaValida.setField("number", "1");  
 entradaValida.setField("pages", "fifth");  
 entradaValida.setField("month", "february");  
 entradaValida.setField("note", "lalala");  
 *assertEquals*(Collections.*singletonList*(entradaValida), teste3);  
}

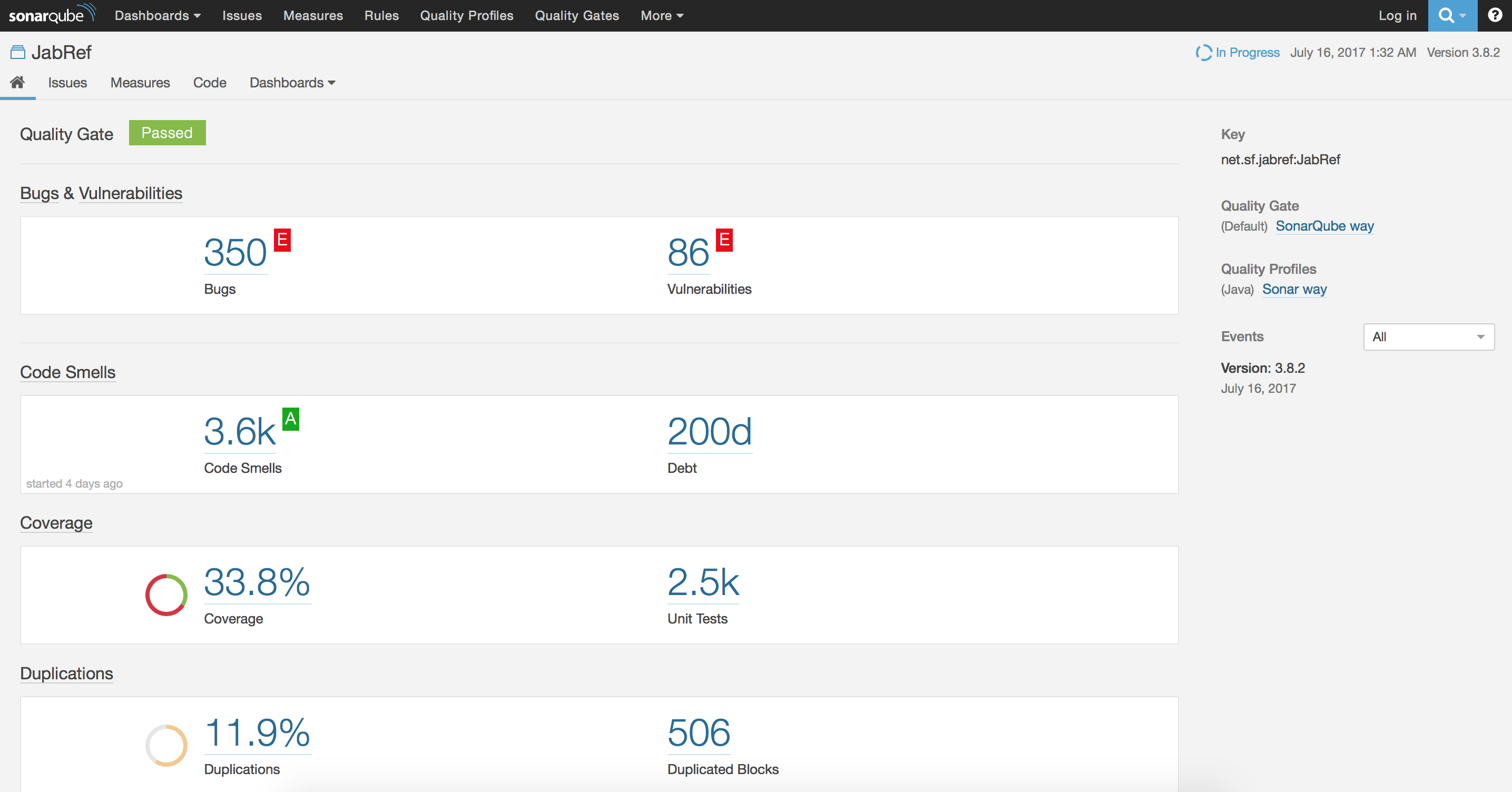
* 1. **Testes Funcionais de importação de um arquivo no database corrente**

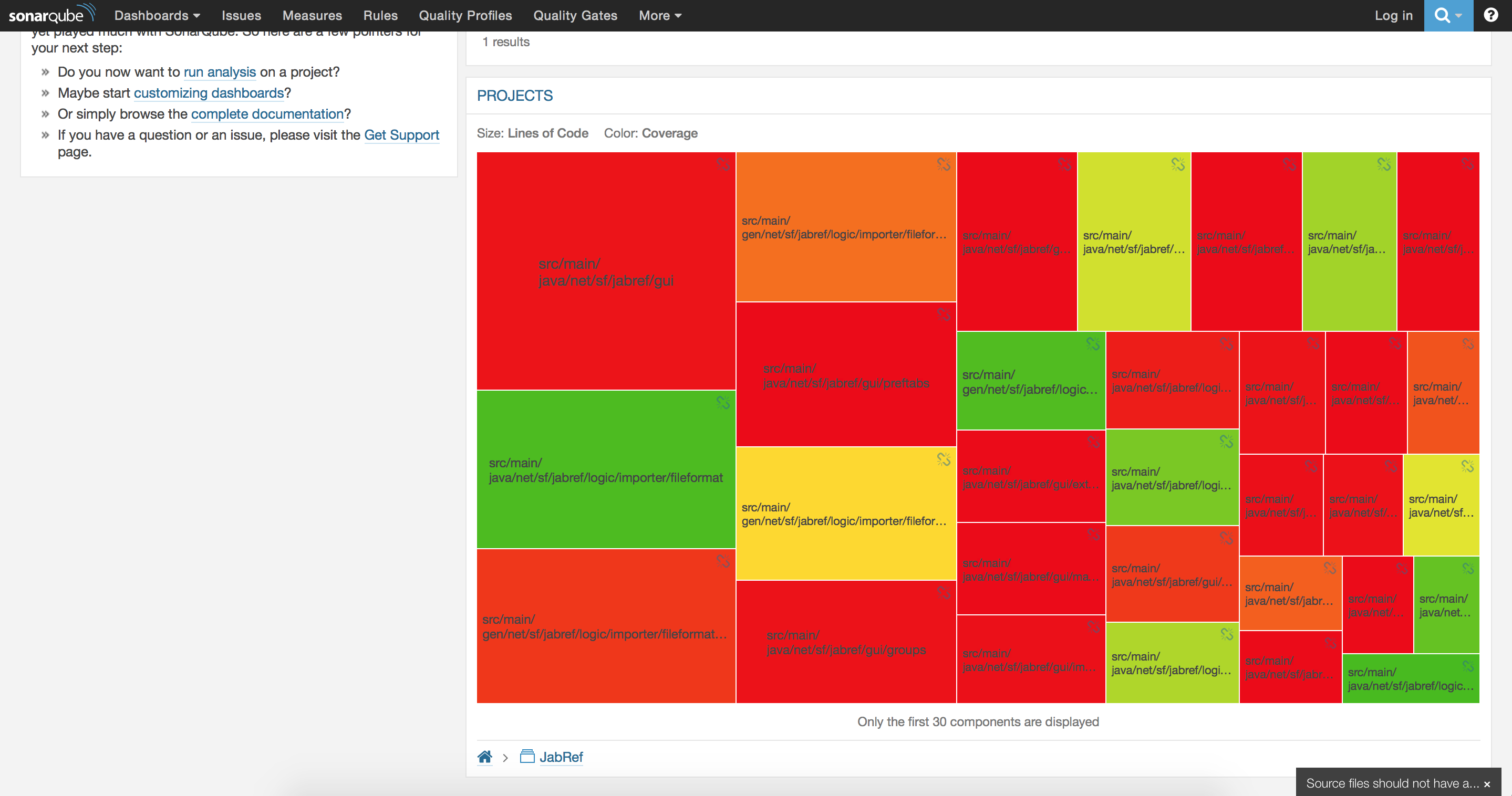
Durante os testes funcionais de importação de um arquivo no database corrente foram criados testes para verificar a importação de arquivos no formato pdf, utilizando para isso um arquivo em branco no mesmo formato, e verificar a importação de um item bibliográfico no formato Bibtex, utilizando para isso valores válidos para os campos necessários.

O código dos testes pode ser verificado abaixo:

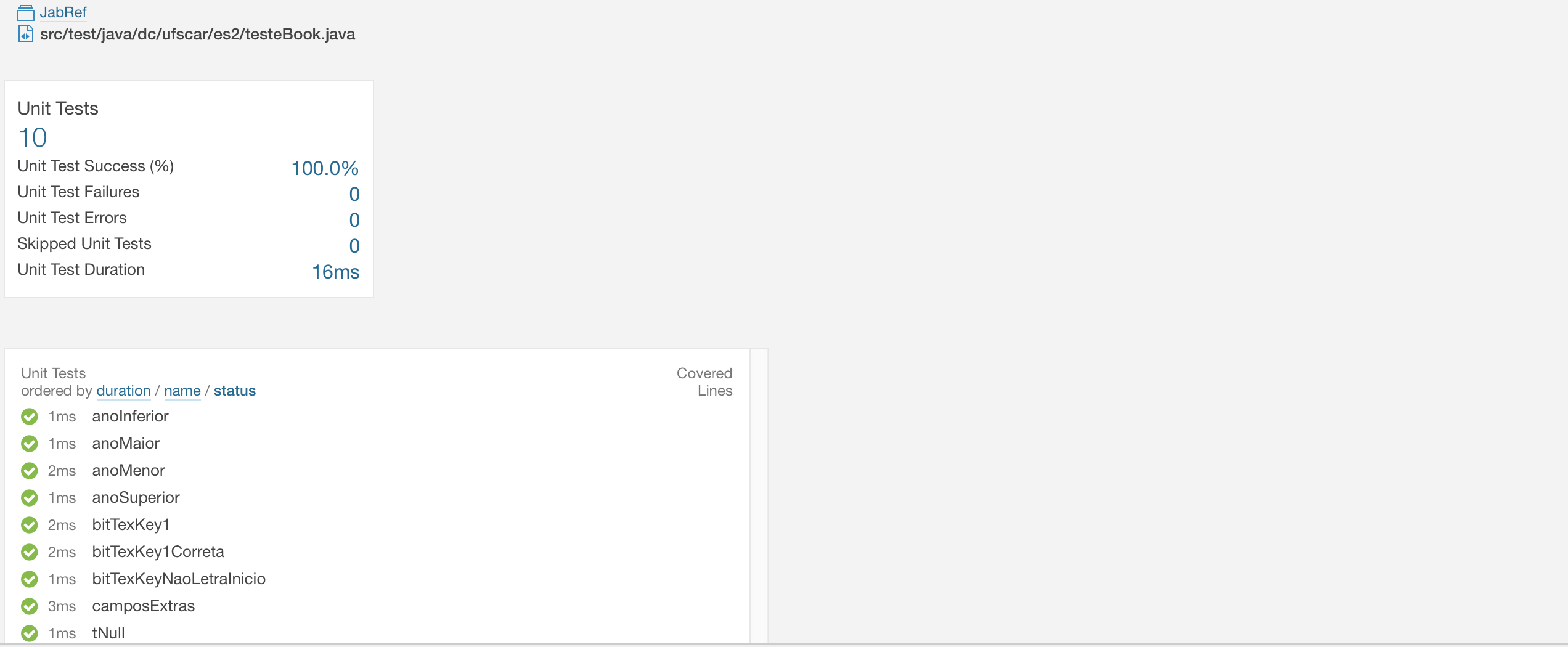
@Test  
public void importPDFTest() throws URISyntaxException {  
  
 PdfXmpImporter importer = new PdfXmpImporter(JabRefPreferences.*getInstance*().getXMPPreferences());  
 Path pdfFile = Paths.*get*(PdfXmpImporterTest.class.getResource("blank.pdf").toURI());  
 List<BibEntry> bibEntries = importer.importDatabase(pdfFile, StandardCharsets.*UTF\_8*).getDatabase().getEntries();  
  
 *assertEquals*(1, bibEntries.size());  
  
 BibEntry be0 = bibEntries.get(0);  
 *assertEquals*(Optional.*empty*(), be0.getField("abstract"));  
 *assertEquals*(Optional.*empty*(), be0.getField("author"));  
 *assertEquals*(Optional.*empty*(), be0.getField("keywords"));  
 *assertEquals*(Optional.*of*("Blank PDF Document"), be0.getField("title"));  
}  
  
@Test  
public void importBibtextTest() throws IOException, URISyntaxException{  
 BibtexImporter importer = new BibtexImporter(JabRefPreferences.*getInstance*().getImportFormatPreferences());  
 Path file = Paths.*get*(BibtexImporterTest.class.getResource("testbib.bib").toURI());  
 List<BibEntry> bibEntries = importer.importDatabase(file, StandardCharsets.*UTF\_8*).getDatabase().getEntries();  
  
 *assertEquals*(2, bibEntries.size());  
  
 for (BibEntry entry : bibEntries){  
 if (entry.getCiteKeyOptional().get().equals("small")) {  
 *assertEquals*(Optional.*of*("Freely, I.P."), entry.getField("author"));  
 *assertEquals*(Optional.*of*("A small paper"), entry.getField("title"));  
 *assertEquals*(Optional.*of*("The journal of small papers"), entry.getField("journal"));  
 *assertEquals*(Optional.*of*("1997"), entry.getField("year"));  
 } else {  
 *assertEquals*(Optional.*of*("Jass, Hugh"), entry.getField("author"));  
 *assertEquals*(Optional.*of*("A big paper"), entry.getField("title"));  
 *assertEquals*(Optional.*of*("The journal of big papers"), entry.getField("journal"));  
 *assertEquals*(Optional.*of*("MCMXCVII"), entry.getField("volume"));  
 }  
 }  
}

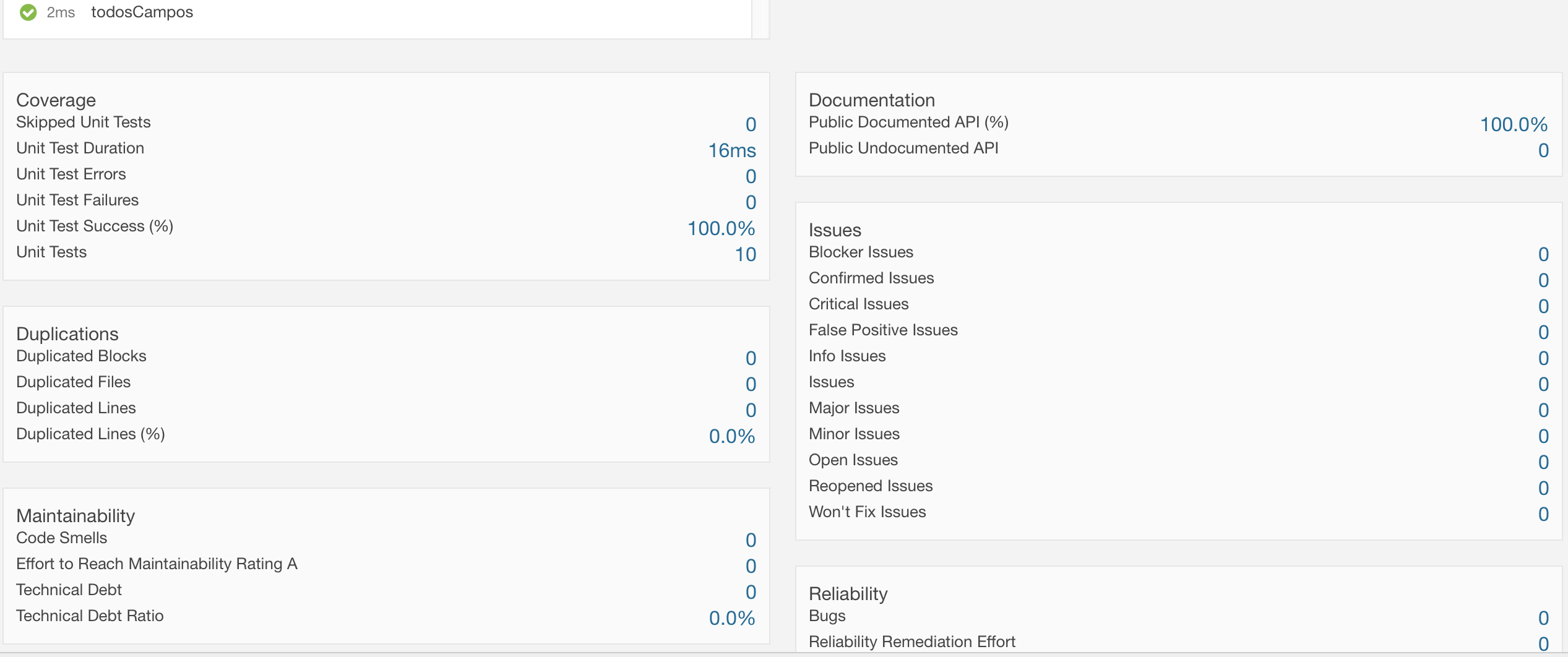
* 1. **Relatório de cobertura de código geral**

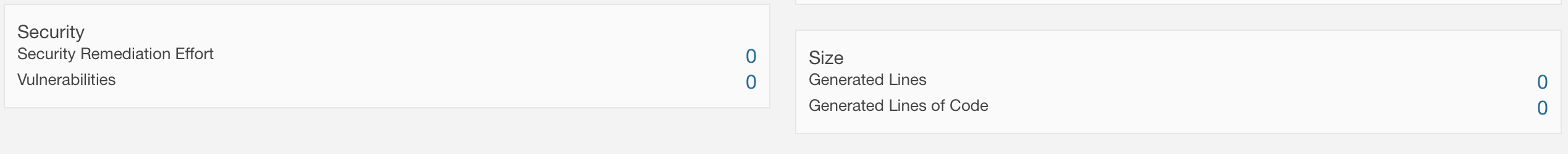




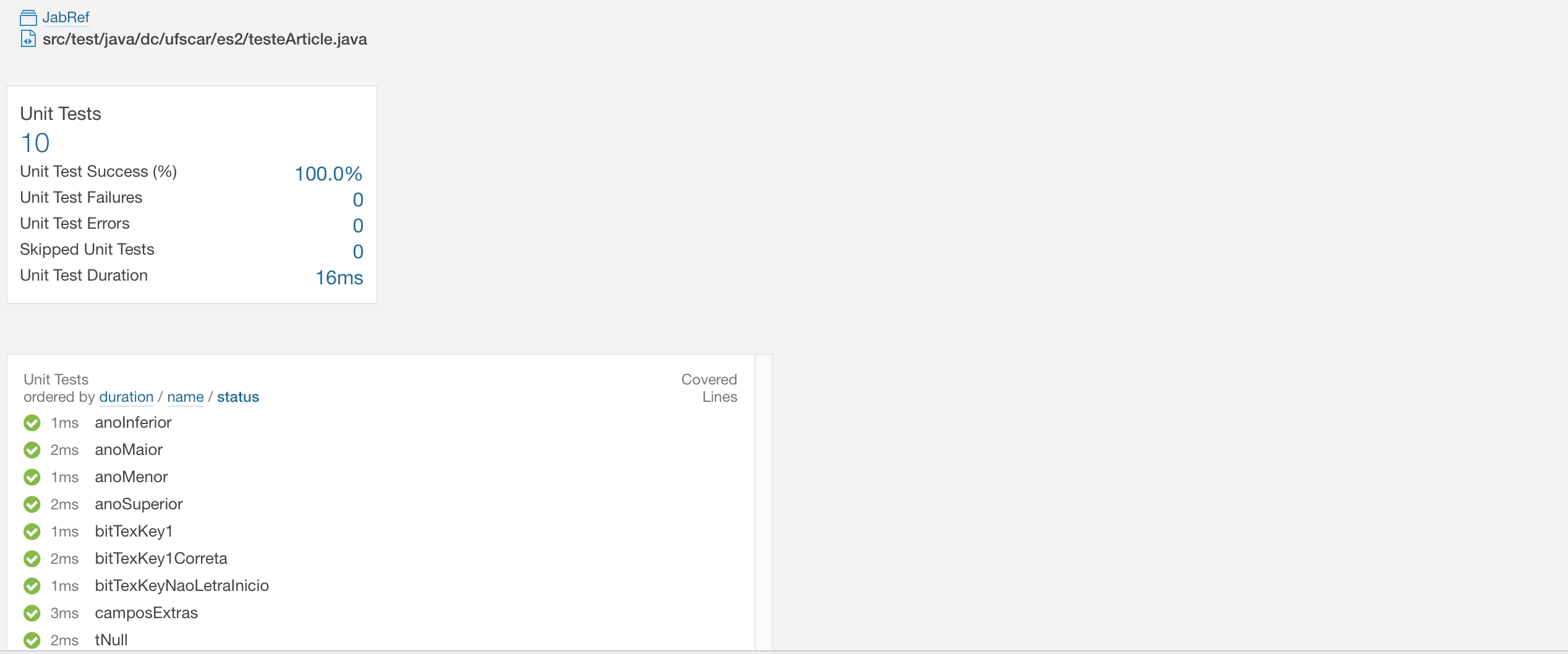
* 1. **Relatório de cobertura de código dos testes funcionais na categoria book**

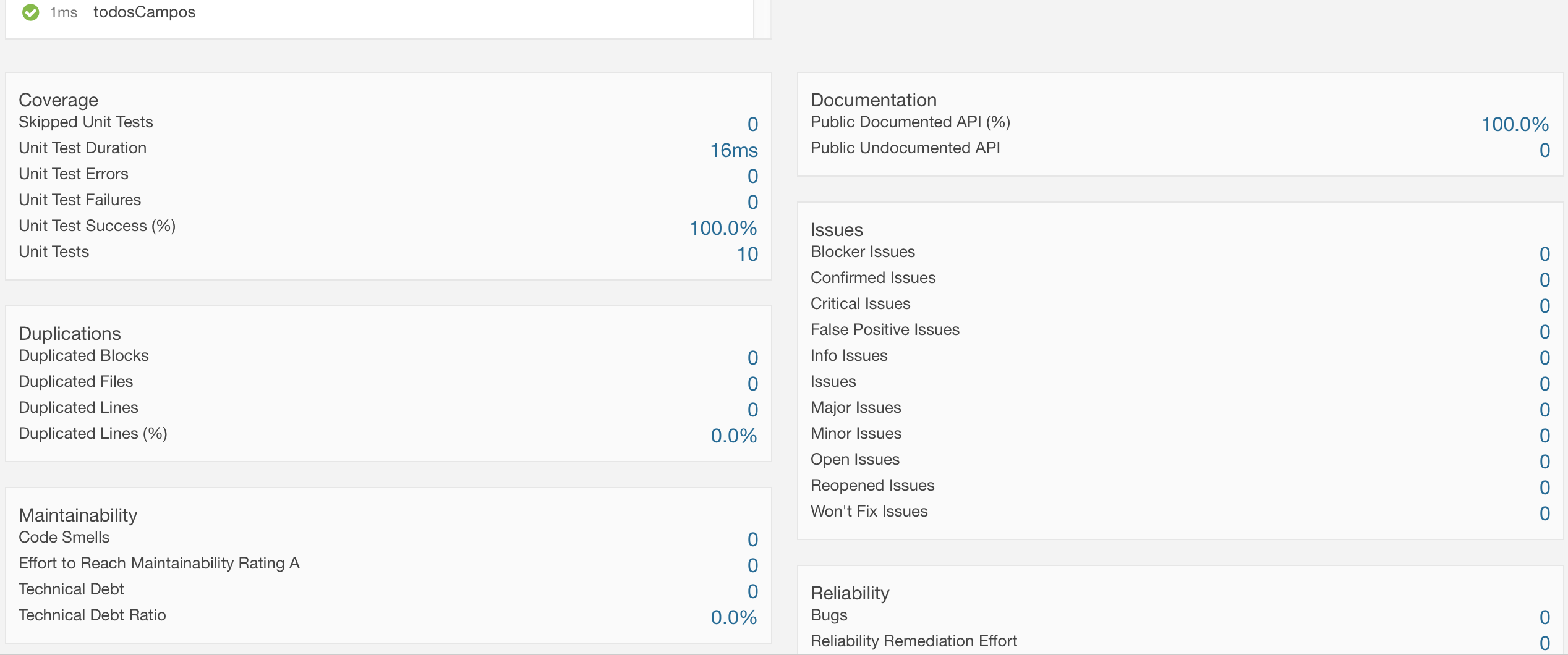


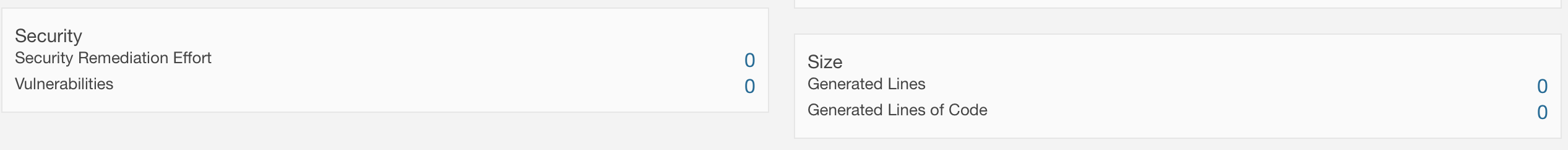




* 1. **Relatório de cobertura de código dos testes funcionais na categoria article**







* 1. **Relatório de cobertura de código dos testes funcionais para importação de arquivos no database corrente**

**ESTÁ FALTANDO GERAR ESSE RELATÓRIO**

**Etapa 2 – Manutenções Perfectivas**

Nessa etapa serão realizadas manutenções para corrigir e melhorar o software do JabRef para isso serão feitas modificações na verificação do campo “year” e do campo “Bibtexkey” de um Bibtex, além de habilitar a possibilidade de importação de arquivos CSV na base corrente.

Para isso, serão criados métodos que farão essas funcionalidades, testes para verificar se os métodos estão agindo dentro do estabelecido e gerar relatórios de cobertura de código para os testes.

O método abaixo foi criado para facilitar a verificação dos campos “year” e “Bibtexkey”.

public String checaCampo(String fieldName, String value){  
 switch (fieldName) {  
 case "year":  
 value = *validaAno*(value);  
 break;  
 case "bibtexkey":  
 value = *validaBibtexKey*(value);  
 break;   
 }  
 return value;  
}

**2.1** **Validação do campo “year”**

Para validar o campo “year” foi estipulado que não poderia ser um valor maior que 2017, pois é o ano que estamos, e não poderia ser um valor menor que 1800, pois é o valor limite que o calendário Java consegue calcular com precisão.

Para isso foi criado o seguinte método:

public static String validaAno (String ano) {  
 String str = null;  
 Calendar calAuxiliar = Calendar.*getInstance*(); //método para instanciar a data do SO  
 int limInferior = 1900;  
 int limSuperior = calAuxiliar.get(Calendar.*YEAR*); //Recebe o ano atual do SO  
 int aux = Integer.*parseInt*(ano); //auxiliar recebera a conversão de ano (string) para int  
  
 //checa validade de data  
 if ((aux < limInferior && aux < limSuperior)){  
 JOptionPane.*showMessageDialog*(null, "Ano inferior ao limite. Tente Novamente!!");  
 return "";  
 }else if (aux > limInferior && aux > limSuperior){  
 JOptionPane.*showMessageDialog*(null, "Ano superior ao limite. Tente Novamente!!");  
 return "";  
 }  
 return ano;  
}

Como podemos ver no código acima, se um valor inválido for inserido o programa deverá mostrar uma mensagem de erro e apagar o valor do campo.

Abaixo temos os testes criados para a categoria book para esse método mas eles também foram criados para a categoria article, além dos testes que já existiam anteriormente:

@Test  
public void anoMaior() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{, year = {2018}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setField("year", "2017");  
 *assertNotEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
@Test  
public void anoMenor() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{, year = {1899}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setField("year", "2017");  
 *assertNotEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
@Test  
public void anoInferior() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{, year = {1900}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setField("year", "1900");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
@Test  
public void anoSuperior() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{, year = {2017}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setField("year", "2016");  
 *assertNotEquals*(Collections.*singletonList*(entradaValida), test);  
}

**2.2** **Validação do campo “Bibtexkey”**

Para validar o campo “Bibtexkey” foi estipulado que não poderia ter menos que 2 caracteres e que o primeiro caracter deverá ser uma letra.

Para isso foi criado o seguinte método:

public static String validaBibtexKey(String name) {  
 if (name.length() < 2) {  
 JOptionPane.*showMessageDialog*(null, "Entrada inválida. Adicione mais caracteres!!");  
 return "";  
 } else if (!Character.*isLetter*(name.charAt(0))) {  
 JOptionPane.*showMessageDialog*(null, "Entrada inválida. Primeiro caracter precisa ser uma letra!!");  
 return "";  
 }  
 else  
 return name;  
}

Como podemos ver no código acima, se um valor inválido for inserido o programa deverá mostrar uma mensagem de erro e apagar o valor do campo.

Abaixo há os testes criados para esse método:

@Test  
public void bitTexKey1() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{a}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setCiteKey("a");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
@Test  
public void bitTexKeyNaoLetraInicio() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{0AAA}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setCiteKey("0AAA");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
@Test  
public void bitTexKeyUmCaracter() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{A}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setCiteKey("A");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test);  
}  
  
  
@Test  
public void bitTexKey1Correta() throws ParseException{  
 List<BibEntry> test = parser.parseEntries("@book{Aaa111}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setCiteKey("Aaa111");  
 *assertEquals*(Collections.*singletonList*(entradaValida), test);  
}

**2.3 Oferecer a opção de importar arquivos CSV**

**COMPLETAR AQUI COM OS CÓDIGOS DOS MÉTODOS**

Abaixo estão os testes criados para esse método:

@Test  
public void testeNomeExtensao() {  
 *assertEquals*("Comma Separated Values", importer.getName());  
}  
  
@Test  
public void testFormatoID() {  
 *assertEquals*("Comma Separated Values", importer.getId());  
}  
  
  
@Test  
public void testsExtensao() {  
 *assertEquals*(FileExtensions.*CSV*, importer.getExtensions());  
}

**2.4 Oferecer a opção de criação de um novo arquivo quando existirem entradas duplicadas**

**COMPLETAR AQUI COM OS CÓDIGOS DOS MÉTODOS E TESTES**

**Etapa 3 – Testes de Regressão**

**COMPLETAR AQUI COM OS CÓDIGOS DOS TESTES, UMA EXPLICAÇÃO E OS TESTES DE COBERTURA**

**Etapa 4 – Manutenção Perfectiva de outra funcionalidade**

Durante essa etapa final foi necessário realizar uma manutenção perfectiva em outra funcionalidade a nossa escolha. A funcionalidade escolhida foi validar o numero de páginas digitadas no campo “pages” quando um Bibtex da categoria Inbook é criado.

Para isso, serão criados métodos, testes e realizados os testes de cobertura.

Para realizar a validação foi necessário alterar um método anteriormente criado para validar o campo “year” e o campo “Bibtexkey”. Ele ficou da seguinte forma:

public String checaCampo(String fieldName, String value){  
 switch (fieldName) {  
 case "year":  
 value = *validaAno*(value);  
 break;  
 case "bibtexkey":  
 value = *validaBibtexKey*(value);  
 break;  
 case "pages":  
 value = *validaNumeroPaginas*(value);  
 break;  
 }  
 return value;  
}

**4.1 Validação do campo “pages”**

Para validar o campo foi preciso estipular que o campo não poderia ser vazio e que não poderia ter um valor menor que 1.

Abaixo temos o código do método:

public static String validaNumeroPaginas(String n\_paginas){  
 int minimo = 1;  
 String empty = "";  
 if(n\_paginas.equals(empty)){  
 JOptionPane.*showMessageDialog*(null, "Número de páginas inválido. Adicione um valor válido");  
 return empty;  
 }else if(Integer.*parseInt*(n\_paginas) < minimo) {  
 JOptionPane.*showMessageDialog*(null, "Número de páginas negativo. Adicione um valor positivo");  
 return empty;  
 }else if(Integer.*parseInt*(n\_paginas) == 0){  
 JOptionPane.*showMessageDialog*(null, "Número de páginas não pode ser igual a 0");  
 return empty;  
 }  
 else  
 return n\_paginas;  
}

Como podemos ver no código acima, se um valor inválido for inserido o programa deverá mostrar uma mensagem de erro e apagar o valor do campo.

Os testes para o método estão abaixo:

@Test  
public void nPaginasNegativo() throws ParseException{  
 List<BibEntry> teste = parser.parseEntries("@inbook{1, title = {A arte da Guerra},\n" + "pages = {-1}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("book");  
 entradaValida.setField("pages", "2");  
 *assertNotEquals*(Collections.*singletonList*(entradaValida), teste);  
}  
  
@Test  
public void nPaginasNulo() throws ParseException{  
 List<BibEntry> teste = parser.parseEntries("@inbook{2, title = {Sherlock},\n" + "pages = {}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("inbook");  
 entradaValida.setField("pages", "3");  
 *assertEquals*(Collections.*singletonList*(entradaValida), teste);  
}  
  
@Test  
public void nPaginasZero() throws ParseException{  
 List<BibEntry> teste = parser.parseEntries("@inbook{2, title = {Harry Potter},\n" + "pages = {0}}");  
 BibEntry entradaValida = new BibEntry();  
 entradaValida.setType("inbook");  
 entradaValida.setField("pages", "2");  
 *assertEquals*(Collections.*singletonList*(entradaValida), teste);  
}

**4.2 Relatório de Cobertura de código do teste de validação do campo “pages”**

**FALTA GERAR ESSE RELATÓRIO**