Pós-Graduação em Ciência da Computação

Ivan Valentim Santos

AVS: Uma ferramenta para mitigação de Duplicação de Relatórios de Erros em empresas de desenvolvimento mobile

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife
2019

Ivan Valentim Santos

AVS: Uma ferramenta para mitigação de Duplicação de Relatórios de Erros em empresas de desenvolvimento mobile

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Recuperação de Informação

Orientador(a): Profº. Dr. Ricardo Bastos Cavalcante Prudêncio

Recife

2019

Catalogação na fonte

Biblioteca Monick Raquel Silvestre da S. Portes, CRB4-1217

S237a Santos, Ivan Valentim

AVS: uma ferramenta para mitigação de duplicação de relatórios de erros em empresas de desenvolvimento mobile / Ivan Valentim Santos. – 2019.

82 f.: il., fig., tab.

Orientador: Ricardo Bastos Cavalcante Prudêncio.

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2019.

Inclui referências.

1. Inteligência artificial. 2. Recuperação da informação. 3. Mineração de texto. I. Prudêncio, Ricardo Bastos Cavalcante (orientador). II. Título.

006.3 CDD (23. ed.) UFPE- MEI 2019-043

Ivan Valentim Santos

“AVS: Uma Ferramenta para Mitigação de Duplicação de Relatórios de Erros em Empresas de Desenvolvimento Mobile”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 20/02/2019.

BANCA EXAMINADORA

Profª. Dra. Bernadette Farias Lóscio (Examinador Interno)
Universidade Federal de Pernambuco

Profº. Dr. Renato Fernandes Corrêa (Examinador Externo)
Universidade Federal de Pernambuco

Profº. Dr. Ricardo Bastos Cavalcante Prudêncio (Orientador)
Universidade Federal de Pernambuco

Dedico a Deus por sempre estar ao meu lado nos momentos mais difíceis desse trabalho e ao meu pai Alfredo Valentim (***in memorian***), que de onde ele estiver, tenho certeza que sempre intercede por mim nos momentos mais difíceis que a vida me impõe.

RESUMO

A adoção de sistemas de gerenciamento de relatórios de erros é algo fundamental em empresas de software durante o processo de produção/teste. Os tipos de informação e a grande quantidade de dados armazenados nesses sistemas levam a desafios que dificultam na produtividade dos processos relacionados à eficiência do rastreamento dos dados, como, por exemplo, a presença de relatórios de erros duplicados. Estudos demonstram que a quantidade de relatórios de erros duplicados pode afetar diretamente na produtividade de uma empresa. Idealmente, um relatório de erro duplicado deve ser identificado antes de ser criado por testadores. Em alguns casos, os gerenciadores de relatórios de erros são apontados como responsáveis por parte desse problema, devido às limitações existentes em seus sistemas de busca. Esta dissertação tem como propósito investigar abordagens e técnicas que possam contribuir para mitigação dos altos índices de relatórios de erros duplicados. Para tal, desenvolvemos a AVS (*Automatic Versatile Search tool*), uma ferramenta de buscas que contribui para a identificação de relatórios de erros com base em técnicas de Recuperação de Informações e Mineração de Texto, visando dar suporte aos GRE (Gerenciadores de Relatórios de erros) para diminuir a ocorrência de relatórios duplicados. Como prova de conceito, implementamos a AVS no contexto do Centro de Teste da Motorola (CTM) no Centro de Informática da UFPE. Toda pesquisa por um relatório de erro candidato a ser aberto é preprocessada. Então, a semelhança entre a busca (*i.e.*, representada por um resumo de um erro candidato a ser um novo relatório) e os relatórios disponíveis no banco de dados é calculada, gerando uma lista de relatórios anteriores ranqueada por similaridade. No final, os relatórios de erros são divididos em grupos, onde seus dados são relacionados de acordo com as semelhanças entre suas sentenças ou palavras chaves, visando produzir um processo mais avançado de identificação de potenciais duplicações. Após um estudo de caso realizado, foi constatada a utilidade da ferramenta principalmente com relação a ganhos de precisão e agilidade do processo de buscas, o que consequentemente, colaborou para melhoraria na produtividade do processo.

Palavras-chave: Mineração de Texto. Duplicação de Relatórios de Erros. Recuperação de Informação.

ABSTRACT

The adoption of error reporting systems is fundamental in software companies during the production/testing process. The types of information and a large amount of data stored in these systems lead to challenges that hamper the productivity of processes related to the efficiency of data crawling, such as the presence of duplicate error reports. Studies show that the amount of duplicate error reporting can directly affect a company's productivity. Ideally, a duplicate error report must be identified before it is created by testers. In some cases, error reporting managers are reported to be responsible for part of this problem, due to limitations in their search engines. This dissertation aims to investigate approaches and techniques that may contribute to the mitigation of the high indexes of duplicate error reports. To do so, we have developed the AVS (Automatic Versatile Search tool), a search tool that contributes to the identification of error reports based on Information Retrieval and Text Mining techniques, in order to support GRE (Bug Tracking Systems) to reduce the occurrence of duplicate reports. As proof of concept, we implemented AVS in the context of the Motorola Test Center (CTM) at the Informatics Center of UFPE. Any search for a candidate error report to be opened is preprocessed. Then the similarity between the search (*i.e.*, represented by a summary of a candidate error being a new report) and the available reports in the database is calculated, generating a list of previous reports ranked by similarity. In the end, error reports are divided into groups, where their data are related according to the similarities between their sentences or keywords, in order to produce a more advanced process of identifying potential duplications. After a case study was carried out, it was verified the usefulness of the tool mainly in relation to the gains of precision and agility of the search process, which consequently collaborated to improve the productivity of the process.

Keywords: Text Mining. Duplication bug Reports. Information Retrieval.

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizagem de Máquina
ANR	Application not responding
AVS	Automatic Versatily Search
BM	Best Matching
GRE	Gerenciadores de Relatórios de erros
IDF	Inverso da frequência nos documentos
LN	Linguagem Natural
MT	Mineração de Textos
PLN	Processamento de Linguagem Natural
RI	Recuperação de Informação
STC	Suffix tree clustering
TF	Frequência do termo
TF-IDF	Frequência do termo – Inverso da frequência nos documentos
VSM	Vector Space Model

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1

INTRODUÇÃO

Com o avanço das tecnologias e a competitividade do mercado, o trabalho constante para fornecer produtos atrativos que conquistem o gosto de clientes é um desafio cada vez mais disputado por empresas de hardware/software.

Para atingir as expectativas quanto a qualidade dos seus produtos para com os seus clientes, as empresas de grande porte precisam de processos de garantia e controle de qualidade, que sejam ágeis e capazes de resolver o máximo de problemas encontrados em um produto durante seu desenvolvimento. De acordo com (BARROS; NETTO; BAIÃO, 2009), os projetos de software geralmente utilizam sistemas Gerenciadores de Relatório de Erros (GRE) como meio de contribuir para garantia de qualidade. Através deles, desenvolvedores, testadores e usuários finais registram as ocorrências de erros de um software e monitoram o progresso de suas correções até que os desenvolvedores finalmente solucionem os problemas.

1.1 MOTIVAÇÃO

Durante o processo de testes de software, constantemente ocorrem submissões de relatórios de erros duplicados. Em boa parte, esse problema ocorre pelo fato dos GRE do mercado ainda serem compostos por sistemas de buscas complexos e limitados. Além disso, a falta de padronização de relatórios e as diversas possibilidades de descrição de um erro (*i.e.*, criadas através do uso da linguagem natural) fazem da duplicação de relatórios de erros algo muito frequente em uma linha de produção. Os relatórios de erros duplicados além de prejudicar na produtividade de testadores e desenvolvedores, geram um acúmulo de informações desnecessárias na base de relatórios de uma empresa, prejudicando também o desempenho dos servidores que armazenam a base. Como consequência disso, o processo de resolução de problemas e erros é afetado, podendo prejudicar diretamente na qualidade final de um produto.

1.2 OBJETIVO GERAL

Seguindo pelo contexto citado, este trabalho se concentrou em estudar uma forma de desenvolver uma ferramenta capaz de contribuir dando suporte à GRE para mitigação dos altos índices de duplicação de relatórios de erros em empresas de software através de buscas mais precisas. Adotando métodos e técnicas de Recuperação de Informação (RI) e Mineração de textos (MT), a solução proposta visa solucionar algumas das limitações encontradas nos GRE atuais, provendo melhores resultados de pesquisa por relatórios.

1.3 OBJETIVOS ESPECÍFICOS

Com base nas áreas de estudo citadas, foi desenvolvida e implementada uma ferramenta chamada AVS, com a principal tarefa de procurar relatórios de erros utilizando abordagens para pré-processamento dos relatórios da base de dados, verificar a semelhança entre suas descrições e uma consulta textual (*i.e.*, definida por um usuário com a intenção de relatar um novo erro), e encontrar padrões entre os dados.

O foco principal é diminuir o percentual de ocorrências de duplicações de relatórios provenientes de buscas imprecisas, ou seja, evitar que relatórios duplicados sejam criados devido às lacunas dos sistemas de buscas dos GRE. Desta forma, como objetivos específicos deste trabalho podemos destacar:

- O uso de Indexação dos relatórios de erros para acesso ágil as informações;
- A aplicação de técnicas de tratamento de texto para estruturação e padronização de conteúdo;
- A criação de tesauros capazes de centralizar informações de um domínio e contribuir para melhores resultados na cobertura e precisão de buscas;
- As melhorias na facilidade de uso do sistema de buscas, considerando as principais dificuldades apresentadas pelas ferramentas atuais;
- O aumento na precisão das buscas a partir de algoritmos de similaridade textual, apresentando os resultados através de uma lista ordenada pelos relatórios anteriores mais relevantes;
- O agrupamento dos resultados em grupos rotulados e divididos pelas semelhanças entre os conteúdos dos documentos, capazes de fornecer descoberta de conhecimento.

Para concretização dos objetivos citados, foi implementada uma ferramenta na linha de produção de uma empresa privada, capaz de atender aos requisitos de acordo com as necessidades e a estruturas fornecidas pelo domínio.

1.4 ORGANIZAÇÃO DO DOCUMENTO

A estrutura deste trabalho é organizada da seguinte maneira:

- **Capítulo 2** apresenta de uma forma geral uma revisão acerca das definições sobre relatórios de erros e GRE. Serão apresentados os principais campos de um relatório, além de seu ciclo de vida durante o processo de produção;
- **Capítulo 3** apresenta a caracterização do problema relacionado a relatórios de erros duplicados, bem como as principais abordagens e estudos relacionados ao tema encontrados na literatura;
- **Capítulo 4** descreve a solução proposta (AVS), detalhando sua arquitetura, suas funcionalidades, sua forma de apresentação dos resultados, e as principais técnicas de tratamento de dados utilizadas;
- **Capítulo 5** descreve um estudo de caso realizado onde a AVS foi comparada com uma ferramenta de linha de produção de testes já conhecida e utilizada em uma empresa privada;
- **Capítulo 6** conclui o trabalho, resumindo os resultados e discutindo propostas para estudos futuros.

2

RELATÓRIOS DE ERROS

2.1 RELATÓRIO DE ERROS

Um relatório de erro (*a.k.a. bug report, issue report, defect report, change request,* etc. (ZHANG et al., 2015)) é considerado um documento que tem como objetivo descrever situações de erros, aprimoramentos, solicitações de mudanças ou problemas em geral que podem ocorrer em um produto de software (CAVALCANTI, 2009).

Os relatórios de erros são vitais durante o ciclo de vida de um projeto, pois permitem que testadores, desenvolvedores e usuários finais (principais responsáveis pela submissão dos relatórios (NGUYEN et al., 2011)) interajam entre si para solucionar problemas encontrados durante o uso de um determinado software (ZIMMERMANN et al., 2010).

Por serem criados por pessoas e para pessoas, geralmente, os relatórios de erros são construídos utilizando a *Linguagem Natural* (LN) (SUREKA; JALOTE, 2010). A figura 1 a seguir demostra um exemplo de um relatório de erros em um projeto de desenvolvimento mobile.

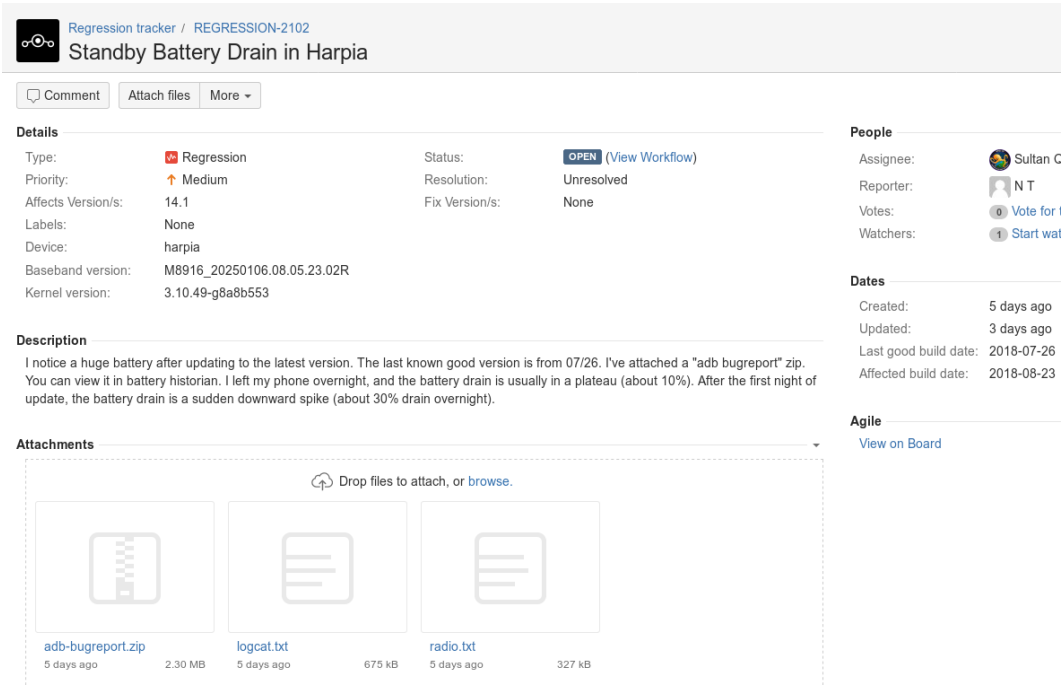

Figura 1 – Exemplo de um relatório de erro - Fonte: (“LineageOS JIRA”, 2018)

Como observado na Figura 1, um relatório de erros normalmente é composto por um número significante de campos (*fields*) de informações. Estes campos contribuem tanto na identificação, quanto no entendimento da situação reportada. Entre os principais campos podem ser destacados:

- ***Summary*** - Campo que intitula o relatório descrevendo resumidamente a situação submetida (erro, mudança, etc.);
- ***Description*** - Campo que descreve os detalhes essenciais para o entendimento do cenário relatado. Geralmente neste campo são informadas: as condições iniciais do ambiente para reprodução da situação (*setup to reproduce*), os passos para sua reprodução (*steps to reproduce*), os resultados observados e esperados (*observed* e *expected results*) (ZHANG et al., 2015), além de *logs* e anexos que detalham o que está sendo reportado (imagens ou vídeos que demonstrem a situação mais detalhadamente). Por fim, neste campo também podem ser anexadas e avaliadas propostas de *patches* de correções (TAMRAWI et al., 2011);
- ***Resolution*** - Campo responsável por indicar o resultado atribuído ao relatório durante ou após o seu tratamento (*e.g.*, *Unresolved*, *Fixed*, *Duplicate*, *Moved*, *Wontfix*, *Cancelled*, etc. (ZIMMERMANN et al., 2010));

- ***Status*** - Campo que descreve o estado do relatório, exemplo: novo (*New*), fechado ou finalizado (*Closed* ou *Resolved*), etc (TAMRAWI et al., 2011).

Além dos citados anteriormente, um relatório deve conter campos que auxiliem na sua priorização e organização, como, tipo (*type*, *e.g.*, *defect*, *task*, etc.), as datas de criação e modificação, nível de gravidade ou importância do relatório (*severity* ou *priority*), versão do software (*software version*), hardware (*product version*), o nome de quem submeteu (*reporter*), o nome de quem está assinado (*assignee*) para cuidar da situação, comentários adicionais, entre outros (CAVALCANTI, 2009; XIA et al., 2014; ZHANG et al., 2015).

No entanto, devido à diversidade de campos oferecidos em sistemas de gerenciamento (GRE) e as variações de domínios estabelecidos a projetos, os campos de um relatório também podem variar. Como resultado disso, cada projeto pode personalizar o *template* do relatório de acordo com as necessidades de seu domínio (CAVALCANTI, 2009).

2.1.1 Ciclo de vida de um relatório de erro

Como um projeto de software é constituído por várias equipes (*e.g.,* desenvolvedores, testadores, responsáveis pela triagem dos relatórios*,* etc.) e dividido em diversos processos, é comum existir um ciclo de vida para os relatórios de erros. Este ciclo de vida pode ser visualizado de duas formas, a partir do ciclo *entre as equipes* e do ciclo de *estados*. Como ilustrado na Figura 2, entre as equipes, o ciclo de vida pode ser representado (de forma genérica) da seguinte maneira.

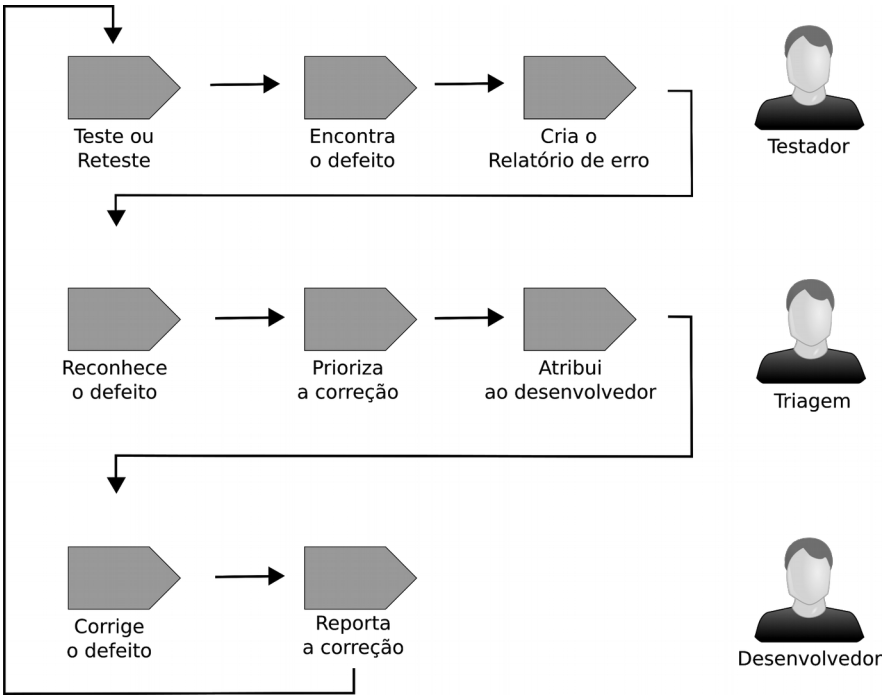

Figura 2 – Ciclo de vida de um relatório de erro entre as equipes de um projeto -
Fonte: Adaptação de (CRISTIANO, 2007)

onde, cada time tem por objetivo realizar tarefas específicas que ajudam a concluir o processo de trabalho em um determinado relatório. Os testadores têm como responsabilidade executar o produto e através disso, procurar situações anômalas, para em seguida reportá-las. Os responsáveis pela triagem dos relatórios tem o trabalho tanto de investigação do seu conteúdo quanto de gerar priorizações e atribuições para o tratamento (*i.e.*, escolha de desenvolvedores para solucionar a situação) (ZHANG et al., 2015). Por fim, os desenvolvedores que são responsáveis pelas correções e pelo encaminhamento das soluções para avaliação (reteste) de testadores.

Outro ponto a cerca do ciclo de vida de um relatório está nas mudanças de estados (*status*), aos quais ele pode ser submetido. Na figura 3, (ZHANG et al., 2015) ilustra os principais estados que um relatório de erro pode ter durante sua vida. Vale ressaltar também, que o modelo dos estados de um relatório pode variar ligeiramente de projeto para projeto (ANVIK; HIEW; MURPHY, 2006).

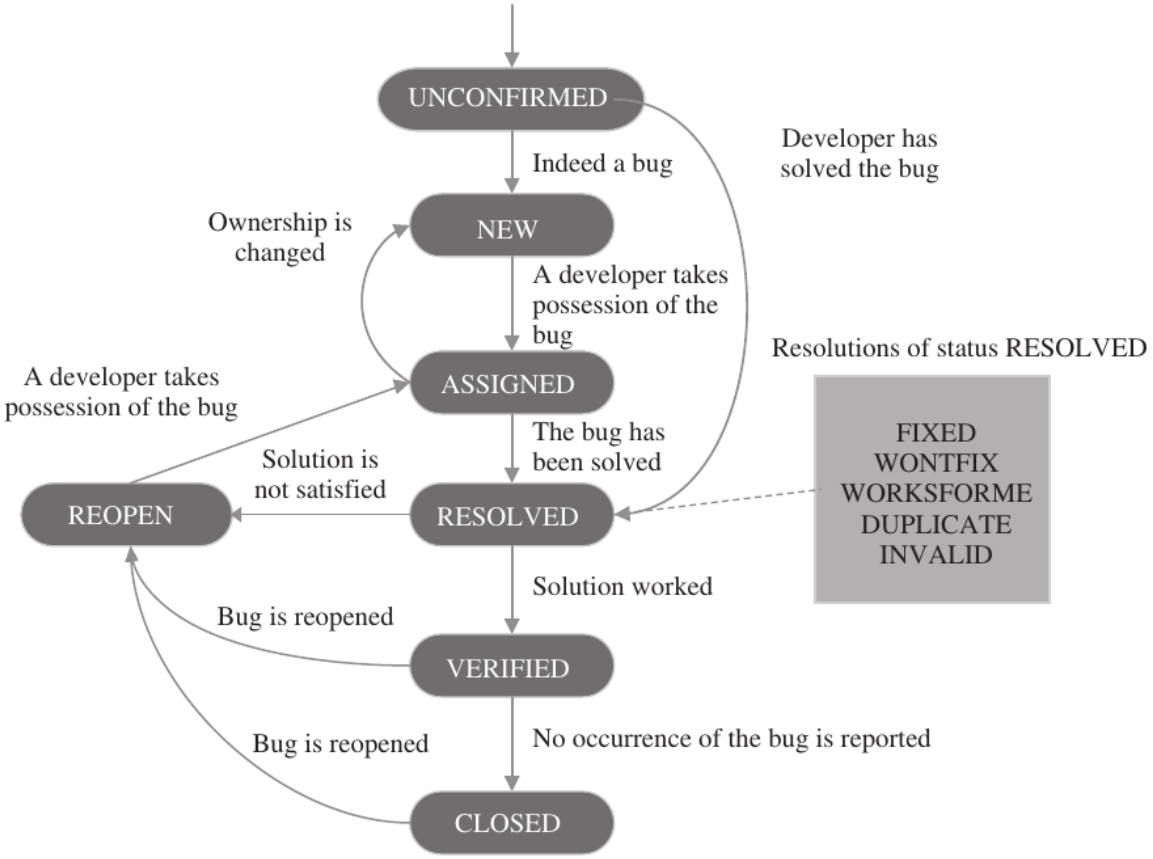

Figura 3 – Ciclo de vida dos estados de um relatório de erro - Fonte: (ZHANG et al., 2015)

De forma genérica, segundo (ANVIK; HIEW; MURPHY, 2006), quando um relatório é aberto, seu estado é definido como Novo (*New*), assim que avaliado pela triagem e confirmado a sua veracidade (*i.e.*, é nesse processo que o relatório pode ser definido como duplicado, inválido ou cancelado), ele é atribuído (*Assigned*) a um desenvolvedor, o qual, após seu tratamento, define o estado final (*e.g.*, *Resolved, Closed, etc.*) atribuindo também uma resolução (*e.g.*, *Fixed, Worksforme, etc.*). No entanto, o relatório também pode ter seu estado modificado para reaberto (*Reopen*) caso o erro ainda ocorra após a sua conclusão.

2.2 GERENCIADORES DE RELATÓRIOS DE ERROS

Segundo (BARROS; NETTO; BAIÃO, 2009), projetos de software frequentemente utilizam sistemas para gerenciamento de relatórios de erros (*a.k.a. Bug Report Systems, Bug tracker, Issue tracker,* etc.), onde gerentes de projeto, desenvolvedores, testadores e usuários finais podem registrar e trabalhar com as ocorrências de erros de um determinado software e consultar o andamento das respectivas correções para garantia de qualidade. O principal

objetivo desses sistemas é facilitar a comunicação entre todos os envolvidos do projeto e ajudar a manter um melhor gerenciamento do tratamento dos erros detectados.

De uma forma geral, esses sistemas têm por função armazenar em uma base de dados todos os relatórios de erros reportados, para que todos os envolvidos no projeto possam ter acesso a esse conteúdo e manipulá-lo, servindo como base de conhecimento. Contudo, os GRE não se resumem apenas a sistemas de buscas (ZIMMERMANN et al., 2009). Neles é possível proporcionar meios de rastreabilidade do processo de correção dos defeitos, através do armazenamento de informações extras e comentários a cerca dos erros durante todo o ciclo de vida do projeto (SUN et al., 2014). Neste caso, o processo de rastreabilidade de erros através do gerenciador de relatórios de erros, ocorre de forma frequente, facilitando o acompanhamento do ciclo de vida de erros, desde o instante em que são identificados até o momento onde suas correções são atribuídas e validadas (BARROS; NETTO; BAIÃO, 2009).

Outra característica muito comum entre os GRE, é a cerca de seus sistemas de armazenamento baseados no modelo relacional de dados (ZIMMERMANN et al., 2009). Isso acontece porque eles trabalham com bases de dados SQL (*e.g.*, MySQL, MS SQL, PostgreSQL, SQL Server, etc.) (MARKO; ALEKSANDAR, 2011). Essa particularidade, por sua vez, permite apenas consultas booleanas baseadas em *SQL* (SUN et al., 2014).

Atualmente na indústria, são muitos os GRE existentes utilizados em grandes projetos. Entre os mais conhecidos podem ser mencionados, o *Jira*[1], o *Bugzilla*[2], o *MantisBT*[3] e o *BugZero*[4].

2.3 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o conceito de um relatório de erro, bem como assuntos relacionados a seu ciclo de vida e seus responsáveis. Além disso, foi abordado o uso dos sistemas de gerenciamento de relatórios, fundamentais para o ciclo de desenvolvimento em um ambiente de produção de software.

1 https://br.atlassian.com/software/jira

2 https://www.bugzilla.org/

3 https://www.mantisbt.org/

4 http://www.websina.com/bugzero/

3

DUPLICIDADE DE RELATÓRIOS DE ERROS

Neste capítulo será apresentada uma breve revisão bibliográfica, contemplando o problema da *duplicidade de relatórios de erros*, bem como as principais abordagens das áreas de *Recuperação de Informação* (RI) e *Aprendizagem de Máquina* (AM), relacionados ao tratamento e manipulação de documentos desse contexto. Além disso, serão descritas as principais técnicas de *Processamento de documentos* utilizadas para o tratamento dos dados. O objetivo desta revisão é pontuar as principais informações relacionadas à detecção de relatórios de erros duplicados que serviram de base para o desenvolvimento deste estudo.

3.1 DEFINIÇÃO DO PROBLEMA

Com as demandas cada vez mais exigentes do mercado em relação a entrega de produtos inovadores em curto tempo, é comum que empresas de desenvolvimento trabalhem progressivamente na melhoraria de sua produtividade utilizando os relatórios de erros.

A submissão de relatórios de erros é fundamental no andamento de um projeto, além disso, é algo que ocorre constantemente durante o ciclo de vida de um software com o intuito de ajudar no processo de correções de erros, provendo assim, mais qualidade a um produto. Entretanto, devido a diversos motivos, é comum que diferentes usuários (*e.g.,* testadores, desenvolvedores, usuários finais, etc.) submetam vários relatórios de erros relacionados a uma mesma situação gerando duplicações (NGUYEN et al., 2012). Entre os principais motivos, podemos citar a quantidade e a distribuição das pessoas envolvidas em todos os processos (SEN; GANPATI; SHARMA, 2014).

Estudos apontam que a quantidade de relatórios duplicados em um projeto pode variar entre valores de 10% à 40% (CAVALCANTI et al., 2013; RUNESON; ALEXANDERSSON; NYHOLM, 2007; SEN; GANPATI; SHARMA, 2014). Esses números demonstram o quanto as duplicações podem afetar diretamente nos resultados da

produtividade, podendo promover de forma negativa, impactos que inevitavelmente prejudicam os processos de manutenção e evolução de um software (CAVALCANTI et al., 2013).

Visto que boa parte das atividades na linha de produção ainda são realizadas de forma manual (RAKHA; SHANG; HASSAN, 2016), esses impactos podem afetar diretamente com:

- Desperdício de tempo e recursos na criação e análise dos relatórios[5] (CAVALCANTI et al., 2013);
- Acréscimo considerável de informações desnecessárias na base de dados do gerenciador de relatórios (ANVIK; HIEW; MURPHY, 2005) (o que também ocasiona em sobrecargas no acesso através do gerenciador de relatórios[6]);
- Aumento do esforço da triagem para uma identificação precisa das duplicações (SWAPNA; THAMMI REDDY, 2016).

Via de regra, como procedimento padrão para submissão de relatórios de erros, quando uma pessoa (*e.g.*, usuário final, testador, etc.) encontra um erro, a ação a ser adotada antes de reportar um novo relatório, é verificar se algum relatório que já descreva o suposto erro não foi aberto anteriormente. Em casos onde haja um relatório aberto, o recomendado é que informações adicionais sejam acrescentadas nos comentários do relatório existente (PRIFTI; BANERJEE; CUKIC, 2011). Porém, nem sempre a tarefa de encontrar um relatório de erro existente para dar seguimento ao procedimento padrão é trivial, pois, como já citado, além da distribuição e da quantidade de pessoas envolvidas em um projeto, vários outros motivos permitem que a duplicidade de relatórios aconteça. Entre esses motivos, podem ser citados:

- O uso da linguagem natural na descrição dos erros (RUNESON; ALEXANDERSSON; NYHOLM, 2007);
- Diferentes níveis de experiência com relação aos erros (RODEGHERO et al., 2016);
- Falta de conhecimento na utilização do gerenciador de relatórios;

5 *e.g.*, quando testadores investem tempo na criação de relatórios que acabam sendo duplicados ou quando relatórios duplicados são atribuídos a desenvolvedores diferentes por engano.

6 Devido o sistema de *buscas booleanas* de um Gerenciador de Relatórios, caso o usuário submeta uma consulta simples (*e.g.*, com poucos termos), o sistema poderá demorar muito processando para encontrar resultados.

- O tipo de busca realizado pelo gerenciador de relatórios (booleana) (BANERJEE; CUKIC; ADJEROH, 2012);
- Um mesmo erro descrito com interpretações ou caminhos diferentes[7] (*i.e.*, passos de reprodução do erro), como ilustrado na Figura 4;
- Entre outros (SWAPNA; THAMMI REDDY, 2016).

A Figura 4 ilustra as duas principais formas de duplicação de relatórios de erros descritas por (RUNESON; ALEXANDERSSON; NYHOLM, 2007).

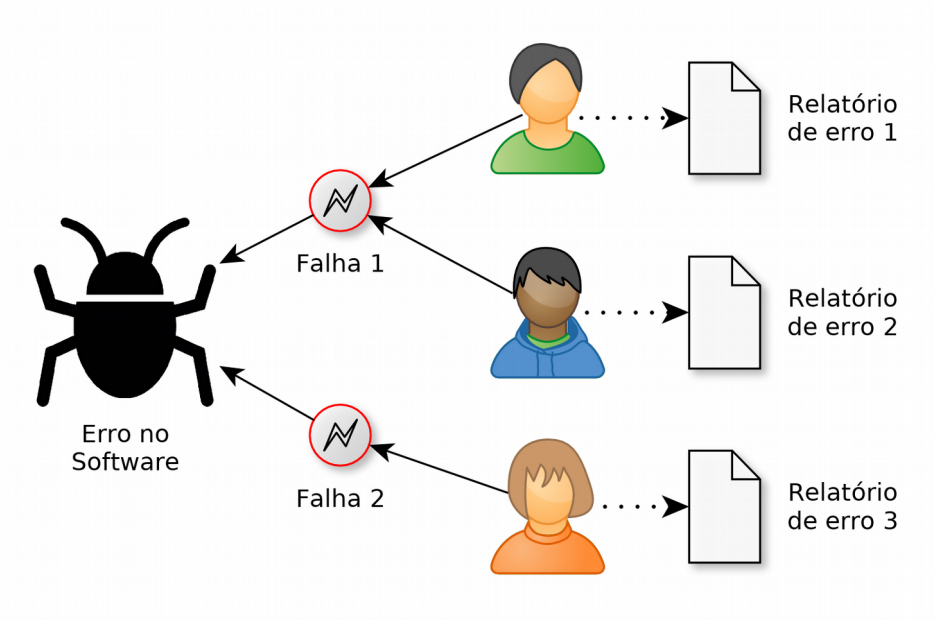

Figura 4 – Principais formas de duplicação de relatórios de erros - Fonte: baseado em (RUNESON; ALEXANDERSSON; NYHOLM, 2007)

Onde:

- Na **Falha 1,** a duplicação é representada por situações onde pessoas diferentes criam mais de um relatório descrevendo o mesmo comportamento para um erro. Neste caso, o *Relatório 2* é duplicata do *Relatório 1* por que ambos descrevem informações obtidas de uma mesma falha (*Falha 1*);
- Na **Falha 2,** a duplicação é representada por situações onde pessoas diferentes criam relatórios descrevendo outros comportamentos observados para um mesmo erro. Situações como a do *Relatório 3* são as mais difíceis de serem descobertas por uma

7 Ocasionalmente, falhas em situações diferentes podem ser causadas por um mesmo erro no código, ou seja, podem ter uma mesma causa raiz.

pessoa que submete o erro, isso porque, os usuários ao encontrar situações diferentes sem ter acesso a uma investigação mais minuciosa (*e.g.* usuários finais reportando um erro), interpretam a situação como um erro novo, criando assim, relatórios que apontam para uma mesma causa raiz (erro) através de uma falha diferente (*Falha 2*).

Neste trabalho, o objetivo fundamental é contribuir para evitar as duplicações de relatórios de erros recorrentes das situações representadas na *Falha 1,* visando com isso, aumentar a eficiência, a produtividade dos processos de teste e manutenção, e consequentemente proporcionar aumento na qualidade de software.

3.2 PRINCIPAIS ABORDAGENS PARA MITIGAÇÃO DE DUPLICAÇÕES

De acordo com (BAEZA-YATES; RIBEIRO-NETO, 2013), a RI é uma área que abrange a Ciência da Computação concentrada sobretudo em dispor para um usuário o acesso fácil e rápido às informações de seu interesse. Neste contexto, (SWAPNA; THAMMI REDDY, 2016) destaca algumas das principais abordagens de RI mais utilizadas na detecção de duplicações de relatórios de erros, entre elas, as *Medidas de Similaridade* e os métodos baseados em Aprendizado de Máquina, como *Classificação* e *Clusterização*. Ainda segundo (SWAPNA; THAMMI REDDY, 2016), elas promovem uma forma eficiente para detectar duplicações de relatórios de erros.

Grande parte dessas abordagens baseadas em RI utilizam sistemas de busca e ranqueamento dos resultados, onde as semelhanças entre relatórios de erros são computadas e listadas de forma ordenada para uma melhor identificação de situações semelhantes (AGGARWAL et al., 2015; CAVALCANTI, 2009; GOPALAN; KRISHNA, 2014; NGUYEN et al., 2012; WANG; LO, 2014).

Na Figura 5, podemos observar os principais processos de uma arquitetura genérica para um mecanismo de buscas por relatórios de erros.

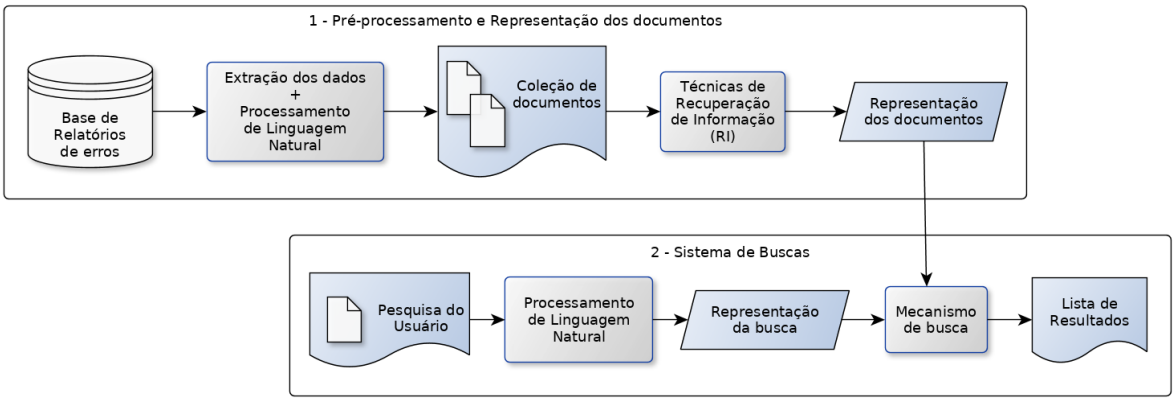

Figura 5 – Arquitetura genérica de Sistemas de busca por relatórios de erros – Fonte: baseado em (SHI; KEUNG; SONG, 2014)

Nessa arquitetura, os principais processos são divididos da seguinte forma:

1. **Pré-processamento e Representação dos documentos**: Nesta fase, os relatórios são coletados de uma base externa e submetidos ao *processamento de linguagem natural,* gerando uma coleção de dados normalizados e estruturados; da coleção de dados, são recuperadas as informações relevantes (Técnicas de RI); ao final do processo, o conteúdo relevante é representado em um formato adaptado para o mecanismo de busca;

2. **Processos do Sistema de buscas**: Assim como os documentos da base, a consulta também é submetida ao *processamento de linguagem natural* e tem seu conteúdo representado em um formato adaptado para o mecanismo de busca; ao final, com a consulta e os documentos, o mecanismo de busca realiza o processo de verificação e apresenta uma lista com os resultados ordenados pela similaridade para serem avaliados pelo usuário.

3.2.1 Abordagens baseadas em Similaridade

As medidas de similaridade são métricas que permitem avaliar quantitativamente a semelhança entre dois itens. A partir da similaridade entre documentos é possível trabalhar com métricas que retornam o quanto um documento é similar a outro. Além disso, os algoritmos de similaridade são fundamentais para os processos utilizados em muitas abordagens de identificação de semelhanças entre documentos.

Ao trabalhar com relatórios de erros e com o intuito de identificar duplicidades, é importante que as informações sejam fornecidas ao usuário em um nível de entendimento

adequado. Para que isso seja possível, alguns estudos apontam a utilização do ranqueamento de relatórios de erros como meio de organização, tanto para uma melhor identificação de semelhanças ou duplicações, quanto para fornecer resultados simples e precisos (BORG et al., 2014; MINH, 2014; WANG et al., 2008). A Figura 6 apresenta um modelo genérico do processo de ordenação por similaridade.

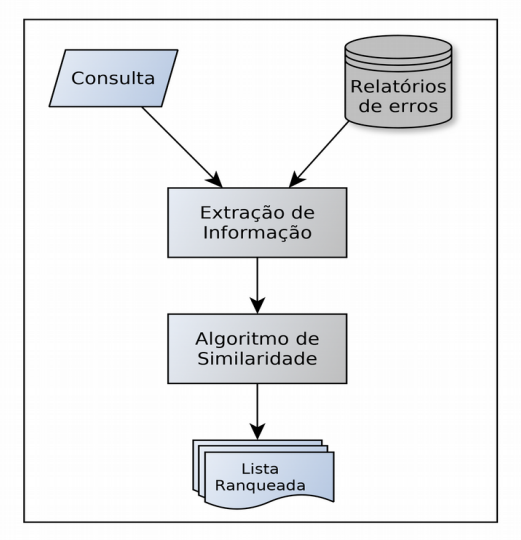

Figura 6 – Processo de ordenação de documentos por similaridade - Fonte: Autor

Um modo de criar uma lista de relatórios de erros ordenados por similaridade é através da representação dos documentos por meio de *vetores numéricos de pesos de termos*, etapa esta que acontece após o pré-processamento dos dados (RUNESON; ALEXANDERSSON; NYHOLM, 2007). Esses valores podem ser extraídos dos vários campos contidos em um relatório de erro para em seguida serem comparados entre si, gerando uma pontuação de similaridade. A partir daí, com o resultado dos cálculos é possível gerar um ranking para apresentar de forma simples a similaridade entre os documentos.

3.2.1.1 Cosseno

O modelo vetorial é uma forma de representação muito usada na recuperação de informações para fornecer um ranqueamento de coleções genéricas. Para calcular o grau de similaridade entre um documento e uma consulta, o modelo vetorial representa ambos como vetores e utiliza o cosseno do ângulo (*i.e.,* similaridade de cosseno) para quantificar a correlação entre os vetores.

Para compensar os efeitos causados pelo tamanho dos documentos no modelo de espaço vetorial, a similaridade de cosseno faz uma ponderação pelo tamanho do documento, padronizando o valor do coeficiente calculado (SILVA; SOUZA, 2013). Esta medida calcula o cosseno do ângulo entre esses vetores (quanto maior o valor do cosseno, maior a similaridade entre os dois vetores) (L. S. GASPAR, 2016). Ou seja, a medida de similaridade mostra o quão próximo os vetores que representam os documentos se aproximam do vetor que representa a consulta.

Assim sendo, um documento d_j e uma consulta de um usuário q podem ser representados como vetores com t dimensões, como é ilustrado na Figura 7.

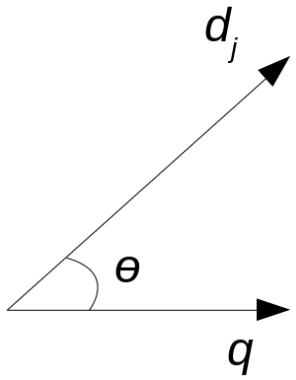

Figura 7 – Representação do cosseno Θ adotado como $sim(d_j,q)$ - Fonte: Autor

No caso de relatórios de erros, por exemplo, a similaridade por cosseno pode considerar como par de vetores, o texto de uma busca realizada e os documentos de um corpus (*e.g.,* relatórios de erros indexados de uma base de dados). Assim, o resultado do cálculo da similaridade entre os vetores indicará a semelhança entre os documentos e uma determinada busca. A fórmula a seguir representa a semelhança entre uma consulta q e um documento d:

$$Similarity(q,d) = \cos\left(\frac{\vec{V}_q \cdot \vec{V}_d}{|\vec{V}_q||\vec{V}_d|}\right) \quad (1)$$

onde $\vec{V}_q \cdot \vec{V}_d$ representa o produto interno dos dois vetores, e $|\vec{V}_q||\vec{V}_d|$ representa o produto da norma dos dois vetores.

Considerado um bom e sólido método básico para ranquear documentos genéricos, o ranqueamento através do modelo vetorial com a similaridade de cosseno é simples e rápido. Isto o faz ser um modelo muito popular e constantemente utilizado para fins de comparação na avaliação de fórmulas alternativas de ranqueamento e de novos modelos de RI (BAEZA-YATES; RIBEIRO-NETO, 2013).

3.2.1.2 BM25

Baseada na teoria da probabilidade, a *BestMatch25* (*a.k.a* BM25 ou Okapi) é uma função de ordenação desenvolvida especialmente para calcular a similaridade entre documentos com diferentes tamanhos. Essa característica de ser sensível às variações de tamanho de documentos faz essa medida ser comumente adotada por sistemas de busca em que as consultas dos usuários geralmente são curtas e consistem em apenas algumas palavras (TIAN; LO; SUN, 2012).

Além do comprimento do documento, o BM25 é capaz de calcular a frequência do termo dentro de um documento e também a frequência de um termo dentro de uma consulta (ROBERTSON; WALKER; BEAULIEU, 2000). Isso ocorre por que o BM25 foi originado para atender a três princípios considerados importantes para uma boa ponderação de termos: a frequência inversa de documentos, a frequência dos termos e a normalização do comprimento dos documentos (BAEZA-YATES; RIBEIRO-NETO, 2013).

Como o modelo probabilístico clássico abrange apenas o primeiro dos princípios, foram feitos vários experimentos para alcançar os outros dois, dando origem ao BM25.

Desde sua criação em 1976 por Stephen E. Robertson (ROBERTSON; JONES, 1976), para dar origem ao BM25 foram feitas algumas mudanças em seu algoritmo. Em sua primeira forma, o BM possui a seguinte fórmula:

$$\frac{(k_3+1).q}{(k_3+q)} \cdot \frac{(k_1+1).f}{(k_1.L+f)} \cdot \log \frac{(r+0.5).(N-n-R+r+0.5)}{(n-r+0.5).(R-r+0.5)} \quad (2)$$

onde:

- k1 e k3 são constantes;
- q é a frequência do termo na consulta;

- f é a frequência do termo no documento;
- n é a quantidade de documentos indexados pelo termo consultado;
- N é o total de documentos na coleção;
- r é a quantidade de documentos relevantes indexados pelo termo consultado;
- R é a quantidade total de documentos relevante;
- L é o tamanho médio dos documentos da coleção;
- 0.5 é uma constante de suavização para lidar com valores de documentos pequenos.

Em seguida Stephen E. Robertson criou o modelo *BM11*, com o intuito de introduzir além da normalização pelo tamanho dos documentos, uma adição de um fator de correção que depende dos tamanhos dos documentos e da consulta. Utilizando o primeiro BM, adicionou um item extra à soma dos pesos dos termos para fornecer a pontuação geral do documento, com a seguinte fórmula:

$$k_2 \cdot n_q \cdot \frac{(1-L)}{(1+L)} \tag{3}$$

onde k_2 é a constante e n_q é a quantidade de termos na consulta (*i.e.,* o tamanho da consulta), permitindo a adição de um fator de correção que depende dos tamanhos tanto dos documentos quanto da consulta. Neste caso, esse fator não depende da correspondência entre os termos da consulta e os termos do documento, apenas dos seus tamanhos (BAEZA-YATES; RIBEIRO-NETO, 2013). Vale notar que quando $L = 0$ o fator completo se torna igual a 1 e não produz efeito no ranking.

Como terceiro passo foi criado o *BM15*, onde foram substituídos ($k_1 \cdot L + f$) no *BM11* por ($k_1 + f$), com isso, foi adicionado um fator para trabalhar com a frequência dos termos.

Por fim, foi elaborada a BM25, com a combinação do *BM11* e do *BM15* adicionando um fator de escala b que pode variar entre valores 0 e 1, onde $b = 0$ reduz a equação a frequência de termos utilizada pelo *BM15* e $b = 1$ reduz para a frequência de termos utilizada no *BM11*. Para os valores dentro dessa escala, o BM25 utiliza uma combinação do *BM11* com o *BM15*. Com isso, a fórmula para o BM25 ficou da seguinte forma:

$$\frac{(k_3+1).q}{(k_3+q)} \cdot \frac{(k_1+1).f}{(K+f)} \cdot \log \frac{(r+0.5).(N-n-R+r+0.5)}{(n-r+0.5).(R-r+0.5)} \quad (4)$$

onde K representa $k_1(b \cdot L + (1 - b))$ [83].

Ao contrário da fórmula original do modelo probabilístico clássico, a fórmula do BM25 pode ser calculada de maneira totalmente automática (*i.e.*, sem depender de informações de relevância provenientes do usuário). Além do mais, muitos estudos afirmam que uma vez que seja bem ajustada, a BM25 fornece resultados melhores que o modelo vetorial na avaliação de coleções genéricas (BAEZA-YATES; RIBEIRO-NETO, 2013). Isso faz com que ela seja adotada em muitos estudos comparativos sobre métodos de ranqueamento, onde substitui o modelo vetorial (L. S. GASPAR, 2016; SHI; KEUNG; SONG, 2014, p. 25).

3.2.2 Abordagens baseadas em Aprendizado de Máquina

Segundo (MURPHY, 2012), a *Aprendizagem de Máquina* (AM) pode ser definida como um conjunto de métodos para detecção de padrões em dados de forma automatizada, com a intenção de prever novos dados baseando-se nesses padrões, além de realizar tomadas de decisão a partir da incerteza. Em outras palavras, AM é uma área da Inteligência Artificial que visa desenvolver técnicas computacionais de aprendizado e sistemas que sejam capazes de adquirir conhecimento automaticamente.

O conceito de AM é dividido em dois tipos: *aprendizado supervisionado*, onde os dados no conjunto de treinamento são seguidos por **labels**[8] (rótulos), identificando a qual classe esses dados pertencem e; *aprendizado não-supervisionado*, onde não há classe pré-definida para o conjunto de dados, assim são realizados conjuntos de observações através dos algoritmos, buscando estabelecer a existência de classes.

3.2.2.1 Classificação

Muito popular e tradicional, a classificação de documentos, também conhecida como *categorização automática*, permite melhorar e facilitar o acesso às informações automatizando processos de identificação e atribuição de documentos de acordo com temas ou assuntos de um domínio definido previamente. Com isso é possível analisar e determinar do que o conteúdo de um documento se trata (MADUREIRA, 2009).

8 *Labels* (Aprendizagem de Máquina) são rótulos utilizados por um conjunto de dados de treinamento e servem para prever os valores dos rótulos de dados de teste não estão rotulados.

Por ser uma subcategoria de aprendizagem *supervisionada* (*i.e.*, a princípio as classes dos documentos são conhecidas), para seu funcionamento um classificador necessita de um treinamento prévio. O treinamento de um classificador pode ser realizado com informações extraídas de documentos. É através do treinamento que são definidas as classes dos documentos e onde os algoritmos aprendem como alocar os dados em cada classe.

A classificação pode ser realizada de duas formas, através da classificação binária de documentos, onde são feitas atribuições de forma que um documento seja considerado de uma classe ou de outra, ou através de multiclasses, onde podem haver mais de uma classe ou tópico de classificação por documento (MCCALLUM, 1999).

São muitos os tipos de algoritmos encontrados em estudos que podem ser utilizados para classificação de documentos, os mais comuns são: as *Redes Neurais*, *Máquinas de Vetor de Suporte* (SVMs), *K-NN* (K vizinhos mais próximos), *Classificadores Naive Bayes, Regressão Logística*, entre outros (ALIPOUR; HINDLE; STROULIA, 2013; CUBRANIC; MURPHY, 2004; LI, 1998; XUAN et al., 2017).

Assim como em outras abordagens, a utilização do pré-processamento textual é uma etapa fundamental para a classificação (ZHOU et al., 2014). O pré-processamento contribui para aumentar a quantidade de palavras relevantes, removendo as palavras com muita frequência ou pouco usadas, ajudando assim a diminuir ruídos e obter resultados mais precisos (CATAE, 2013).

Na área de relatórios de erros a classificação é utilizada como meio de automatizar e agilizar o processo de identificação de algumas situações. Baseando-se no aprendizado com dados de relatórios de erros de uma base, um classificador pode ajudar na tomada de decisões precisas. Por exemplo, indicar se um relatório de erro é realmente um erro ou não (SOHRAWARDI; AZAM; HOSAIN, 2014; ZHOU et al., 2014), apontar possíveis duplicações entre relatórios de erros (AGGARWAL et al., 2015), indicar níveis de severidade, etc. Essa abordagem contribui principalmente para diminuir o trabalho manual exercido por times de triagem (XUAN et al., 2017). A Figura 8 mostra de forma genérica o procedimento de classificação de relatórios de erros.

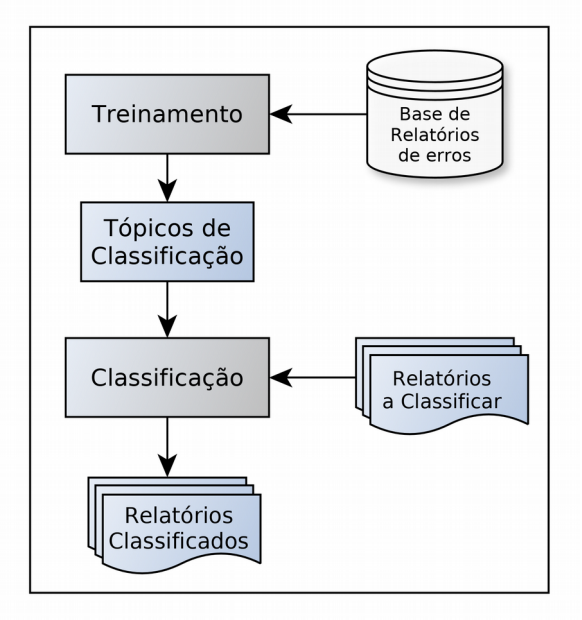

Figura 8 – Processo de Classificação de relatórios de erros - Fonte: Autor

Contudo, a classificação ainda apresenta muitos desafios e dificuldades. Um deles é a complexidade de trabalhar com a semântica e os conceitos abstratos da linguagem natural de textos (LI, 1998). Outro ponto é a consistência e a qualidade dos dados de treinamento que deve ser suficiente para evitar problemas como sobreajuste (*overfitting*), onde dados de treinamento com alto nível de ruídos provocam um excesso de ajustes do classificador sobre a amostra de dados levando a perda de desempenho e imprecisão de resultados (SILVA JÚNIOR, 2016).

3.2.2.2 Agrupamento

Outra abordagem associada à aprendizagem de máquinas é o agrupamento de dados. Também conhecido como clusterização, o agrupamento de dados é definido como o processo de encontrar e rotular grupos de objetos semelhantes em conjuntos de dados (AGGARWAL; ZHAI, 2012). De uma forma geral, a clusterização corresponde ao processo de agrupar automaticamente os elementos (documentos) de uma coleção de dados (corpus) que possuam alguma propriedade em comum. Esses grupos formados, também conhecidos como rótulos, devem apresentar um resultado onde os elementos de um rótulo tenham maior similaridade entre si enquanto tem menor similaridade com elementos de outros rótulos (OCHI; DIAS; SOARES, 2004).

Diferente do processo de classificação, o agrupamento de dados é uma subcategoria da aprendizagem *não supervisionada*, ou seja, as técnicas de aprendizado são aplicadas num conjunto de dados em que não se conhece previamente as suas possíveis classes. Sendo assim, a técnica de agrupamento de texto não requer rótulos no treinamento prévio (como no caso da classificação) para realização do processo (LIMSETTHO et al., 2014). Desta forma, ao trabalhar com agrupamentos de textos é possível identificar padrões e obter Descoberta de Conhecimento a partir de uma coleção de documentos.

Os métodos de agrupamento podem ser considerados rígidos (*hard*) ou flexíveis (*soft*) de acordo com o padrão pertencente de um objeto, isto é, o método é rígido caso o objeto pertença exatamente a um único rótulo ou flexível caso pertença a vários rótulos com diferentes graus (PANDA; SAHOO; PATNAIK, 2013).

Ao processar relatórios de erros, assim como a classificação, o uso agrupamento visa prover uma automatização da tarefa que costumeiramente é feita de forma manual por times de triagem. Através dele é possível contribuir tanto para diminuição do percentual de relatórios de erros duplicados (DANG et al., 2012; GOPALAN; KRISHNA, 2014; JALBERT; WEIMER, 2008; MINH, 2014), quanto para classificar e categorizar erros (LIMSETTHO et al., 2014, 2016; NAGWANI; VERMA, 2011).

A principal vantagem apresentada nesta abordagem é a possibilidade de identificar áreas que vão além das pré-definidas para classificação. Além disso, devido não necessitar de treinamento supervisionado, a quantidade de esforço humano necessária na rotulagem é minimizada. Por fim, ainda é possível encontrar características e pontos desconhecidos na estrutura dos relatórios de erros (LIMSETTHO et al., 2014).

A Figura 9 mostra de forma genérica o procedimento de agrupamento de relatórios de erros.

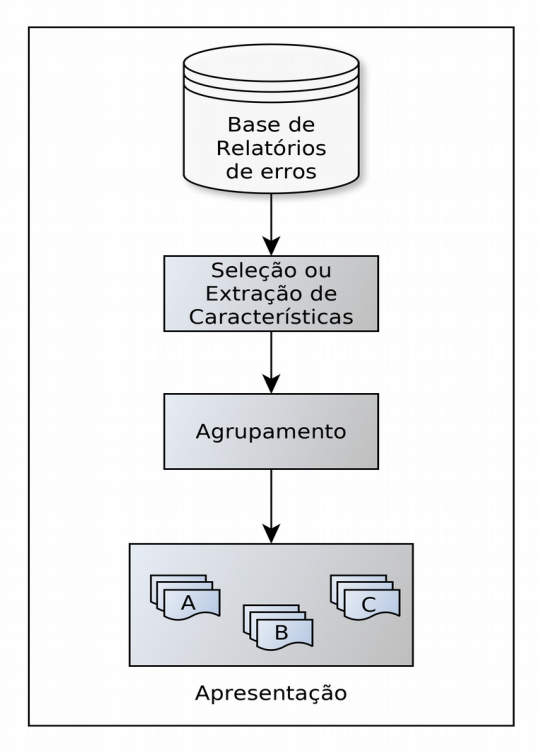

Figura 9 – Processo de Agrupamento de relatórios de erros - Fonte: Autor

Entretanto, o agrupamento de dados também tem suas dificuldades ao manipular conjuntos de dados. Por existirem inúmeras técnicas de agrupamento, é difícil encontrar uma técnica geral em que todas as variedades de estruturas que podem estar presentes em um conjunto de dados sejam encontradas (COSTA, 1999), principalmente porque os dados dificilmente estarão estruturados de uma forma ideal para manipulação. Além disso, cada novo algoritmo de agrupamento criado pode apresentar um comportamento superior aos já existentes (MOSCATO; VON ZUBEN, 2002).

3.3 REPRESENTAÇÃO DOS DADOS

Ao se lidar com documentos textuais como relatórios de erros, uma etapa fundamental é a adequação do texto a um formato que possa ser reconhecido por algoritmos de mineração (SCHIESSL, 2009). Para isso, na área de RI, existem três modelos clássicos que são comumente utilizados para representação de dados, o *Booleano*, o *Vetorial* e o *Probabilístico*.

3.3.1 Modelo Booleano

Os primeiros sistemas de RI eram baseados na representação booleana de recuperação de informação. No modelo booleano os documentos e consultas são representados como conjuntos de termos de índices, permitindo aos usuários especificar suas buscas utilizando uma combinação de operadores lógicos (*AND*, *OR* e *NOT*) para obter documentos que satisfaçam as suas necessidades.

Por ter como base a Álgebra Booleana e a Teoria dos Conjuntos, no modelo booleano toda consulta realizada é interpretada como uma expressão lógica (*true* ou *false*), impondo pesos binários as resultados. Neste caso, a relevância de documentos é obtida a partir da satisfação a uma expressão lógica, ou seja, um documento é considerado relevante caso atenda os termos de lógica da consulta (SOARES, 2009). A fórmula a seguir representa o processo booleano de busca entre uma consulta q e um documento d_j:

$$sim(d_j,q) = \begin{cases} 1 \rightarrow se\ sim(d_j,q) = 1,\ documento\ (d_j)\ é\ relevante\ à\ consulta\ (q); \\ 0 \rightarrow se\ sim(d_j,q) = 0,\ documento\ (d_j)\ não\ é\ relevante\ à\ consulta\ (q); \end{cases} \quad (5)$$

Apesar de ser considerado um modelo com excelente performance e de fácil implementação, os sistemas booleanos apresentam várias deficiências. Por exemplo, embora os resultados retornados sejam correspondentes em alguma ordem, como por data, por ordem alfabética ou algum outro ponto de um documento, ainda assim é notória a ausência de uma indicação de relevância entre os resultados (*i.e.*, ordenação por relevância dos dados). Além disso, outro ponto negativo é a complexidade que um usuário tem para formar uma boa sintaxe de pesquisa (INDURKHYA; DAMERAU; FRED, 2010).

Com exceção dos problemas conhecidos que demonstram que os sistemas booleanos são menos eficazes do que outros sistemas de recuperação, muitos usuários avançados ainda usam sistemas booleanos por se sentirem com mais controle do processo de recuperação (SINGHAL, 2001).

3.3.2 Modelo Vetorial

No Modelo de Espaço Vetorial (VSM – *Vector Space Model*), diferente da representação booleana, os documentos são representados através de *vetores de termos*. Cada termo em um documento é associado a um peso numérico (*i.e.* não binário) que indica a importância do termo para o documento em relação a um corpus (coleção de documentos)

(LESKOVEC; RAJARAMAN; ULLMAN, 2014). Em cada vetor, a ordem das palavras nos documentos não é levada em consideração gerando assim uma representação chamada de sacos de palavras (*bag-of-words*). Neste caso, todas as palavras em cada saco são contadas e o número de vezes que cada palavra aparece é armazenado em um vetor (*i.e.,* termo-por-documento) (MANNING; RAGHAVAN; SCHÜTZE, 2008).

Devido à atribuição de pesos não binários aos termos, tanto das consultas como dos documentos, o VSM é muito utilizado para calcular a similaridade entre documentos armazenados em um sistema e uma consulta de um usuário. Ao considerar o casamento parcial entre cada documento e os termos da consulta, o VSM pode ordenar os resultados de forma decrescente de acordo com o grau de similaridade (BAEZA-YATES; RIBEIRO-NETO, 2013). Quando comparado ao modelo booleano, o principal efeito proporcionado pelo VSM é o aproveitamento mais preciso na apresentação dos resultados fornecidos para o usuário.

Uma técnica comumente usada para definição de pesos é a TF-IDF (*term frequency-inverse document frequency*) (STEIN; DE BARCELOS SILVA, 2016). O TF-IDF é calculado através da multiplicação do valor de $tf_{i,j}$ (número de ocorrências de um termo i em um documento f), pelo df_i (número total de documentos (corpus) que contém i), ou seja,

$$tfidf_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right) \tag{6}$$

onde N é o número total de documentos (corpus).

Uma característica fundamental do modelo vetorial é permitir que várias técnicas de recuperação de informação possam ser utilizadas, entre elas: indexação, agrupamento (clusterização), classificação, etc (SOUCY; MINEAU, 2005; XU; LIU; GONG, 2003). Além disso, o VSM tem como principais vantagens: seu esquema de ponderação de termos que melhora a qualidade da recuperação; sua estratégia de casamento parcial que permite uma recuperação de documentos que se aproximam as condições da consulta; a possibilidade de utilização da fórmula de cosseno para calcular o grau de similaridade; e a normalização pelo tamanho do documento que vem naturalmente embutida no modelo (BAEZA-YATES; RIBEIRO-NETO, 2013).

Contudo, vale observar que o modelo de espaço vetorial pode apresentar uma desvantagem em sua representação quando um documento é muito maior que outro. Por

representar a importância de cada termo em um documento através de vetores resultantes, o modelo vetorial pode indicar que dois documentos são muito similares mesmo eles sendo significativamente bem diferentes. Isso ocorre, por que os valores absolutos do peso podem ser distintos mesmo quando há uma distribuição idêntica em ambos os documentos (SILVA; SOUZA, 2013).

3.3.3 Modelo probabilístico

A modelagem probabilística tem como função classificar documentos em ordem decrescente de relevância probabilística de acordo com uma consulta estabelecida por um usuário (BONFIM, 2009). Isso deve-se ao fato deste modelo ser baseado no princípio probabilístico de ordenação. A ideia fundamental deste princípio é que, para cada consulta ao sistema, deverá existir um conjunto de documentos que contenham exatamente os documentos relevantes (*i.e.,* conjunto de respostas ideal) e nenhum outro (CORREIA, 2018).

O princípio probabilístico de ordenação é definido da seguinte forma: Dada uma consulta q e um documento d_j de uma coleção, o modelo probabilístico tenta estimar a probabilidade do usuário achar um documento d_j de seu interesse (*i.e.,* relevante a sua consulta). Para o modelo, se supõe que essa probabilidade de relevância depende apenas das representações da consulta e do documento, isto é, das informações disponibilizadas ao sistema. Além do mais, o modelo também supõe que deva existir um conjunto de todos os documentos que o usuário deseja como conjunto resposta para a sua consulta. Este conjunto resposta ideal é chamado de R e deve conter documentos identificados como relevantes à consulta, já os documentos fora desse conjunto são definidos como não relevantes (BAEZA-YATES; RIBEIRO-NETO, 2013).

Entretanto, segundo (BAEZA-YATES; RIBEIRO-NETO, 2013), este princípio é problemático, pois, a relevância para o usuário pode ser afetada por fatores externos ao sistema. Com isso, o conjunto resposta ideal pode não atender o que o usuário necessita. Outro ponto, é que o princípio não diz de uma forma clara como são calculadas as probabilidades de relevância nem como é dado o espaço amostral que define as propriedades do conjunto ideal.

No modelo probabilístico, um documento d_j é representado por um vetor de pesos binários que indicam a presença ou ausência de termos, como veremos na fórmula a seguir.

$$\vec{d}_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j}) \quad (7)$$

onde, $w_{i,j} = 1$ se o termo k_i ocorre no documento d_j e $w_{i,j} = 0$ caso não ocorra.

Para calcular a similaridade, se supõe que R é um conjunto de documentos inicialmente estimados como relevantes para uma consulta q. Seja $\overline{R}$ complemento de R (*i.e.,* conjunto de documentos não relevantes). $P(R|\vec{d}_j, q)$ é a probabilidade de que o documento $\vec{d}_j$ seja relevante para a consulta q. Além disso, $P(\overline{R}|\vec{d}_j, q)$ é a probabilidade de que o documento d_j não seja relevante para a consulta q. Com isso, a similaridade entre d_j e q é definida pela seguinte fórmula:

$$Similarity(d_j, q) = \frac{P(R|\vec{d}_j, q)}{P(\overline{R}|\vec{d}_j, q)} \quad (8)$$

O modelo probabilístico tem como principal vantagem o princípio probabilístico de ordenação, que uma vez garantido, resulta em um ótimo comportamento do método (CARDOSO, 2004). Contudo, uma das principais desvantagens, é que, na prática, este comportamento depende da precisão das estimativas de probabilidade, sobretudo para avaliar a relevância e não-relevância dos documentos, principalmente porque a relevância de um documento é afetada por variáveis externas ao sistema. Além do mais, mesmo com a presença do componente IDF, o método não explora a frequência dos termos (TF – *Term Frequency*) no documento (fazendo com que todos os pesos sejam binários). Outras desvantagens nesse modelo são a necessidade de estimar separadamente os documentos em conjuntos relevantes e não relevantes, e a ausência de normalização pelo tamanho dos documentos (BAEZA-YATES; RIBEIRO-NETO, 2013). Esses problemas são solucionados através do modelo BM25, que será discutido mais à frente.

3.4 PRÉ-PROCESSAMENTO

No estudo de (SWAPNA; THAMMI REDDY, 2016) é recomendado como passo inicial a utilização de técnicas de PLN (Processamento de Linguagem Natural), cujo uso é fundamental para manipulação, limpeza e estruturação dos dados (RUNESON; ALEXANDERSSON; NYHOLM, 2007). A fase de pré-processamento utiliza as técnicas de PLN para uma melhor manipulação dos dados textuais. Visto que o conteúdo de um relatório de erro é composto quase que totalmente por informações textuais (podem também existir anexos como imagens, vídeos, logs, etc.), faz-se necessário lidar com caracteres especiais,

pontuações, números, palavras comuns (*stopwords*), verbos conjugados, sinônimos, entre outros itens da linguagem natural (RUNESON; ALEXANDERSSON; NYHOLM, 2007).

O objetivo desta etapa é preparar o conteúdo dos documentos para que o processo de extração de conhecimento possa ser mais efetivo. De uma forma geral, o pré-processamento é um processo semiautomático. Isso se dá devido à dependência da intervenção humana para conduzir a identificação de problemas presentes nos dados. Além disso, para melhoria dos resultados, é necessário investigar quais métodos são mais apropriados para solucionar cada um dos problemas (BATISTA, 2003).

Segundo (BAEZA-YATES; RIBEIRO-NETO, 2013), o pré-processamento de documentos pode ser dividido em cinco processos essenciais para transformação textual: a *tokenização*, a *eliminação de stopwords*, o procedimento de *stemming de palavras*, a *indexação* do conteúdo e a utilização de *tesauros*. A Figura 10 ilustra um fluxo dos principais processos utilizados para pré-processamento de relatórios de erros.

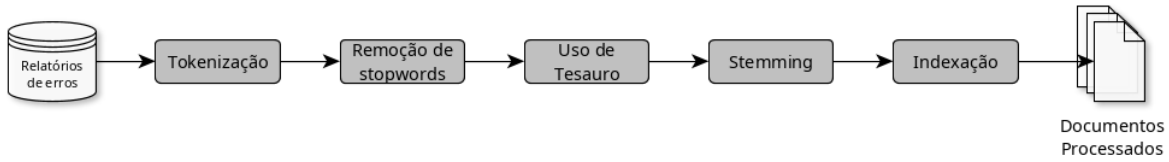

Figura 10 – Principais etapas do pré-processamento textual de um relatório de erro - Fonte: Autor

As etapas do pré-processamento são iniciados assim que as informações são coletadas e organizadas na forma de um conjunto de dados (BATISTA, 2003). Para dar início a normalização dos documentos, é necessário a remoção de pontuações e caracteres especiais, como também, eliminar a sensibilidade entre letras maiúsculas e minúsculas (*case sensitive*) (CORREIA, 2018). A intenção por trás desses processos é padronizar o conteúdo para que ele possa ser manipulado de forma adequada, contribuindo para o reconhecimento preciso de seu significado. A seguir, serão apresentados com mais detalhes cada um dos principais processos.

3.4.1 Indexação

Como a quantidade de documentos em uma base de dados pode se tornar muito grande e de difícil controle, é necessário que haja uma forma ágil e eficaz para realização de

buscas mais precisas por informações em documentos. Para isso ser possível é comum a utilização do processo de indexação.

A função da indexação é selecionar termos (*i.e.* palavras-chave) de um documento em função de seu conteúdo, de modo a representar um índice, e incluir o documento num determinado repositório de informações onde o seu acesso será realizado mais rapidamente (CORREIA, 2018). A principal forma de indexação utilizada atualmente é através do índice invertido, onde o repositório de informações consiste num conjunto de palavras-chave no qual, para cada termo, é guardada uma lista de documentos em que este ocorre. Com os documentos indexados, ao fazer uma busca por um termo, é possível localizar imediatamente no repositório todos os documentos que contêm a incidência daquele determinado termo (BAEZA-YATES; RIBEIRO-NETO, 2013).

A Figura 11 demonstra as principais etapas do processo de indexação, são elas: a definição do vocabulário de indexação, a atribuição de termos de índice a cada documento, e a construção da estrutura de dados do índice (CORREIA, 2018).

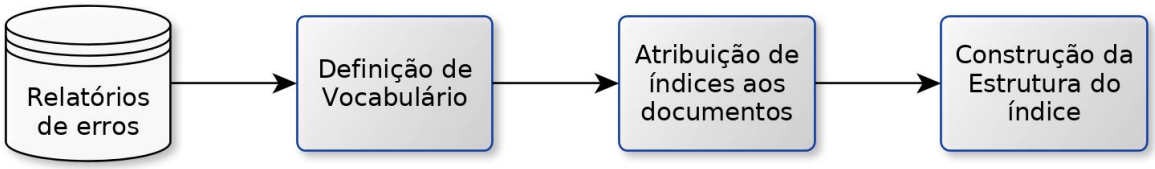

Figura 11 – Etapas do processo de Indexação - Fonte: Autor

No entanto, para o processo de indexação funcionar de forma precisa, são necessários alguns cuidados relacionados ao tratamento dos dados. Pois, se realizado de forma exaustiva, o índice de cobertura aumenta, e consequentemente a precisão na busca pelos documentos diminui (CORREIA, 2018). Este problema pode ser minimizado através do uso de uma outra técnica da recuperação da informação, a remoção de *stopwords*, que ajuda no processo de eliminação de termos irrelevantes, o que proporciona redução de dados desnecessários (BORGES, 2009). Essa e outras técnicas serão descritas nas próximas subseções.

3.4.2 Tokenização

O processo de tokenização consiste em transformar um texto em um conjunto de *tokens*, onde basicamente, cada *token* é uma palavra. Neste caso uma palavra pode ser

definida como uma cadeia de caracteres alfanuméricos, um conjunto de palavras podem ser separados por delimitadores que como espaços, sinais de pontuação, etc (TIAN et al., 2015). Nesta etapa também se inclui a filtragem de todos os símbolos sem importância para o texto, como pontuações, caracteres especiais, etc. A necessidade deste processo se dá principalmente porque esses caracteres não contribuem para a manipulação do conteúdo. Além disso, é recomendável que durante o processo que todos os caracteres em letras maiúsculas sejam substituídos pelos correspondentes em caixa baixa, para assim garantir uma padronização do conteúdo (LAMKANFI et al., 2010). Na Figura 12, é apresentado um exemplo do processo de tokenização.

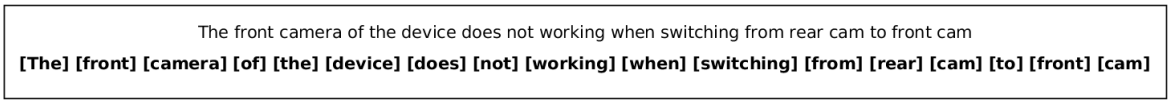

Figura 12 – Processo de tokenização de uma frase - Fonte: Autor

Como o “espaço” é considerado um delimitador é comum ser sempre descartado. Contudo, é importante ressaltar que em algumas línguas não se utiliza o espaço como delimitador, por exemplo, as línguas japonesa e chinesa. Neste caso, (BRITO, 2017) aconselha a utilização de duas abordagens: uma para as línguas em que o espaço é o delimitador e outra para aquelas que não utilizam o espaço como delimitador.

3.4.3 Eliminação de palavras irrelevantes

Após a etapa de tokenização é comum que sejam removidas palavras que têm pouca ou nenhuma relevância no texto, essas palavras são chamadas de *stopwords*. As *stopwords* podem incluir pronomes como "*it*", "*we*" e "*you*", verbos de ligação como "*is*", "*am*" e "*are*", etc. Com este processo de remoção, é possível reduzir o tamanho da estrutura de indexação dos documentos de forma considerável, chegando a alcançar mais de 40% somente com a redução dos termos irrelevantes (BAEZA-YATES; RIBEIRO-NETO, 2013).

Contudo, como os projetos variam de domínios é necessário observar a forma de utilização dessa técnica. Se usada incorretamente a remoção de *stopwords* pode prejudicar o resultado do processamento dos dados, removendo palavras que tenham importância para a interpretação de um erro (ZAMAN; MATSAKIS; BROWN, 2011). Para se evitar problemas neste processo é recomendável investigar previamente o domínio do projeto e observar quais palavras devem compor a lista de palavras irrelevantes (*stoplist*). Desta forma, é possível

garantir que nenhuma palavra relevante seja removida e que palavras irrelevantes não escapem do processo. Um exemplo disso está em situações onde verbos como "*close*" e "*take*", adjetivos como "*on*" ou "*off*" (*e.g.* "*camera ins't on*"), podem ter importância para descrever como determinadas ações ou estados revelam um erro. Na Figura 13, é apresentado um exemplo do processo de remoção de *stopwords* após o processo de tokenização.

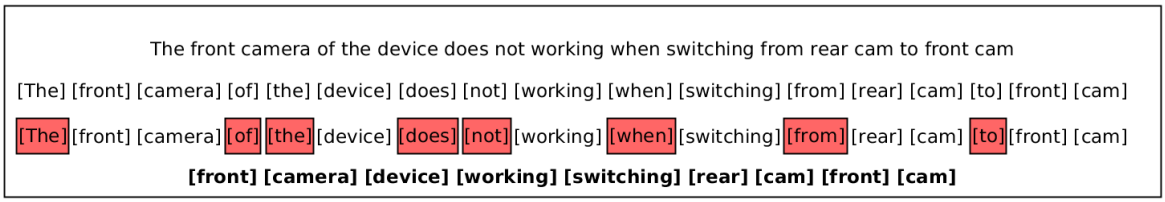

Figura 13 – Processo de eliminação de stopwords - Fonte: Autor

3.4.4 Tesauro

Em um domínio que envolve linguagem natural, como o de relatórios de erros, é comum que pessoas descrevam situações semelhantes de formas diferentes. Para solucionar problemas relacionados às formas distintas que determinados termos podem ter em um relatório, é recomendada a utilização de tesauros (*thesaurus*) (RUNESON; ALEXANDERSSON; NYHOLM, 2007).

Para (BAEZA-YATES; RIBEIRO-NETO, 2013), um tesauro é definido como uma lista pré-compilada de palavras importantes em um determinado domínio de conhecimento, onde, para cada palavra dessa lista, há um conjunto de palavras relacionadas. Este processo é importante para lidar com documentos como relatórios de erros, onde é comum uma pessoa relatar uma funcionalidade, um nome de uma aplicação ou até mesmo os passos para execução de uma situação de uma forma diferente da outra.

De acordo com (ABREU, 2010), em grande parte dos textos escritos por pessoas ligadas ao mundo digital é comum que usuários "brinquem" com as normas e regras ortográficas de um idioma, inovando com construções criativas e inusitadas que abandonam os aspectos formais da ortografia. Assim sendo, um texto digital não pode ser caracterizado como "uma forma errada de escrita", e sim como algo escrito de forma intencional. Ainda segundo (ABREU, 2010), o objetivo disto, é adaptar a linguagem de comunicação a um contexto onde o importante é economizar tempo na escrita real juntando elementos característicos da fala com elementos característicos da escrita, mantendo a informalidade e

criando uma linguagem para identificação de termos digitais. Nesses casos é frequente o uso de abreviações, neologismos, etc, por exemplo, "*cam*", "*camera*" e "*photographic camera*" são situações aparentemente distintas, porém, com o mesmo significado.

Devido esses fatores, é necessário a criação de uma lista de termos relacionados tanto por sinonímia quanto por neologia e redução através de siglas que possa contribuir auxiliando uma pessoa na realização de consultas onde algo pode ser descrito de formas diferentes.

Para facilitar o processo de identificação de sinônimos, no estudo de (KEVIC et al., 2013), é utilizada a NHunspell[9], uma biblioteca C# de código aberto que proporciona verificação ortográfica, hifenização e tesauro. Além dela outra alternativa é o WordNet[10]. Contudo, a principal desvantagem do uso de tesauros está na diversidade de termos específicos entre domínios, pois, a criação de um tesauro específico para cada projeto de forma manual é um processo muito demorado. Para ajudar na solução deste problema, existem vários estudos ligados a essa área (LAGUTINA et al., 2015; TIAN; LO; LAWALL, 2014; VIRGINIA; NGUYEN, 2011), todavia, dependendo da complexidade envolvida em um domínio, ainda pode existir a necessidade de coleta de sinônimos, neologismos, siglas e termos equivalentes de forma manual. Na Figura 14, é apresentado um exemplo do processo de utilização de tesauro após o processo de remoção de *stopwords*.

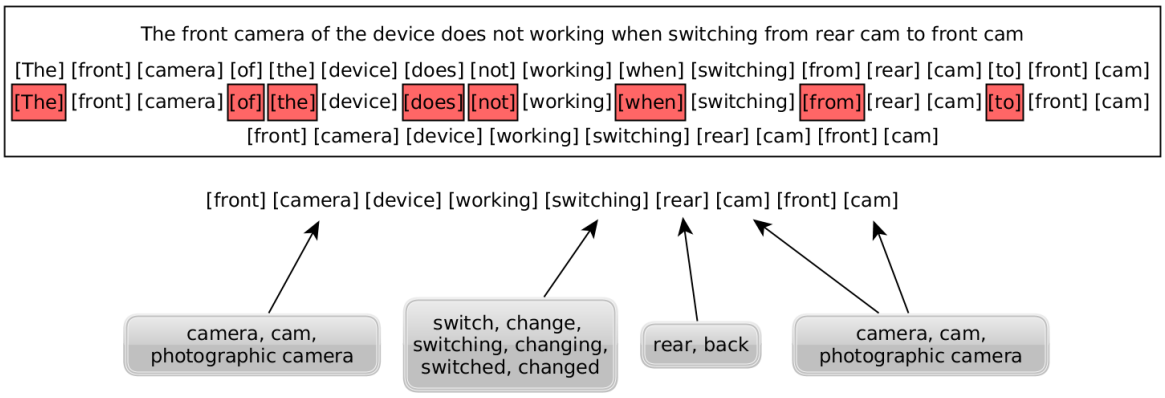

Figura 14 – Processo de tesauro - Fonte: Autor

Apesar de aumentar a precisão e a revocação, um cuidado a ser ressaltado com o uso de tesauros é a necessidade de se controlar o nível de expansão e qualidade de seu

9 https://sourceforge.net/projects/nhunspell/

10 https://wordnet.princeton.edu/

vocabulário. Isso ocorre devido ao aumento de possibilidades entre as sentenças, o que retorna uma quantidade maior de documentos (podendo aumentar o número de documentos irrelevantes), possibilitando com isso, a diminuição de precisão nos resultados (CORREIA, 2018).

3.4.5 Stemização

Em textos produzidos com linguagem natural, várias são as regras gramaticais que podem ser empregadas para modificar o radical de uma palavra, entre elas, singular, plural, presente, pretérito, futuro, etc. Essas variações são conhecidas como variantes morfológicas (DE ÁVILA; SOARES, 2013). Em uma palavra o radical é o morfema que contém o seu significado básico. Neste caso as palavras originárias de um mesmo radical, não são idênticas entre si, porém, são semanticamente relacionadas.

Como essas situações são muito frequentes em relatórios de erros, é fundamental empregar o uso de stemização (truncagem) para trabalhar com as derivações de palavras do texto. A stemização é uma técnica capaz de reduzir uma palavra à sua forma radical, limpando os afixos que carregam informação gramatical ou lexical sobre a palavra (MORAL et al., 2014). Por exemplo, na situação onde o verbo "trabalhar" em inglês são comuns as seguintes conjugações: "*works*", "*working*" e "*worked*", ao realizar o processo de stemização, todas as palavras são reduzidas a "work". No processamento de relatórios de erro, esta técnica é fundamental para padronização de seu conteúdo, evitando que conjugações de uma palavra criem diferenças no entendimento de sua descrição. Além disso, a técnica de stemização contribui para melhorar a performance e a eficiência das pesquisas (CORREIA, 2018).

O algoritmo mais conhecido e utilizado para stemização do conteúdo de relatórios de erros é o *Porter* (CAVALCANTI, 2009; JALBERT; WEIMER, 2008; WANG; LO, 2014; WEN; WU; CHEUNG, 2016), porém, outros algoritmos também podem ser utilizados para a mesma tarefa (MORAL et al., 2014). Além disso, é importante destacar que dependendo do domínio do projeto, os algoritmos de stemização podem variar quanto à precisão (SOHRAWARDI; AZAM; HOSAIN, 2014), nestes casos é interessante estudar previamente os efeitos causados pelo processo para selecionar o melhor algoritmo. Na Figura 15, é apresentado um exemplo do processo de stemização após o processo de tesauro.

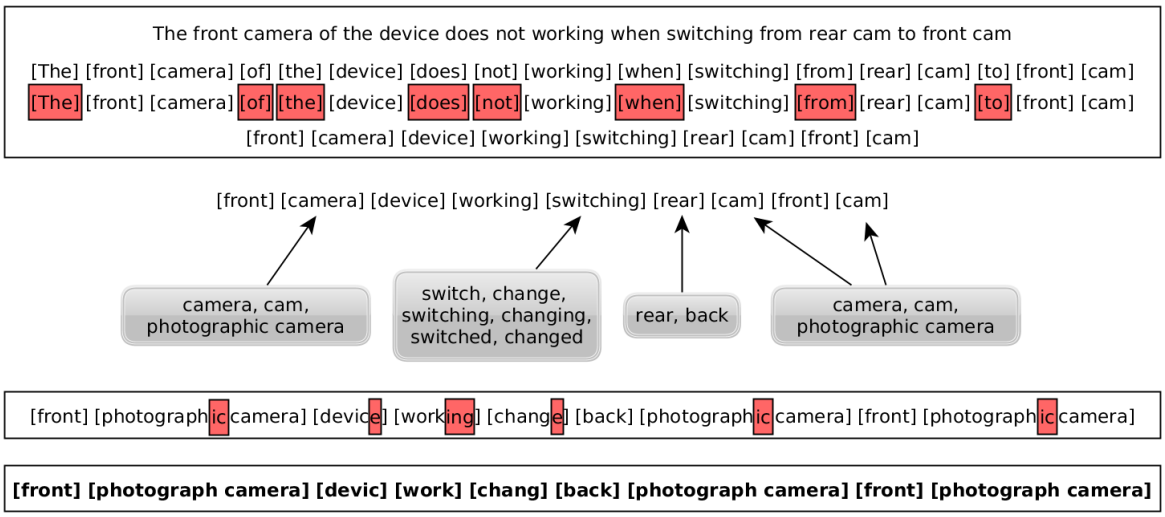

Figura 15 – Processo de stemização - Fonte: Autor

Estudos anteriores realizados por (ROCHA et al., 2015, 2016; SWAPNA; THAMMI REDDY, 2016; ZOU et al., 2016), demonstram a importância do uso das técnicas de PLN apresentando resultados favoráveis, entretanto, em (HINDLE, 2016), é ressaltado que dependendo do projeto, o uso de algumas técnicas de PLN podem não prover resultados significativos.

Para evitar possíveis problemas é importante salientar que para se obter resultados precisos, o uso das técnicas de PLN deve ser devidamente avaliado e configurado. Além disso, é indispensável considerar que de acordo com o domínio do projeto, é recomendável que haja um estudo quanto a adoção das técnicas de PLN para que se encaixem de forma adequada às suas necessidades (L. S. GASPAR, 2016). Ou seja, é necessário investigar quais técnicas são mais adequadas e se as que são utilizadas estão convenientemente ajustadas para tratar os dados de forma correta. Com isso, é possível evitar resultados imprecisos devido à perda de informações importantes (SAHA et al., 2013).

3.5 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado um estudo onde analisamos a problemática relacionada a duplicação de relatórios de erros, as principais abordagens e as técnicas mais utilizadas na literatura para manipulação de relatórios de erros em um ambiente de produção de testes de software.

Para cada uma das abordagens apresentadas é essencial que haja a preparação dos dados que serão trabalhados. O uso de *Pré-processamento* e *Análise Textual*, contribui de

forma significativa para uniformização do conteúdo, ou seja, padronizar os dados para uma melhor coleta de informações importantes de cada relatório de erro. Com isso é possível garantir que os resultados sejam mais precisos e relevantes.

4

TRABALHO DESENVOLVIDO: AVS

4.1 INTRODUÇÃO

Como visto anteriormente, o problema de duplicação de relatórios de erros é algo comum e que gera problemas significantes para um projeto de software. Sobretudo, quando são levados em consideração o acúmulo de relatórios com informações duplicadas e o tempo desperdiçado na criação e análise desses relatórios.

Neste capítulo, é apresentada uma ferramenta chamada AVS (*Automatic Versatily Search*), construída para agilizar e facilitar a atividade de buscas e análise de relatórios de erros com maior precisão, objetivando dar suporte aos GRE na detecção e mitigação de possíveis duplicações entre relatórios antes que eles sejam criados.

O restante deste capítulo é organizado da seguinte forma: 4.2 apresenta sua arquitetura; 4.3 explica o processo de coleta e indexação da base de relatórios; 4.4 aborda o pré-processamento dos dados empregado; 4.5 indica a similaridade utilizada; 4.6 demonstra o processo de agrupamento dos resultados; 4.7 apresenta a interface de usuário da AVS; e por fim, na subseção 4.8 as considerações finais do capítulo.

4.2 ARQUITETURA

O intuito da AVS é prover um sistema de buscas por relatórios de erros capaz de fornecer resultados mais precisos, evitando perda de informações relevantes e facilitando o acesso a informações duplicadas contidas em um domínio que frequentemente utiliza linguagem natural. Para isso ser possível, sua estrutura foi criada a partir de 2 ferramentas de código aberto, o *Apache Solr*[11], um famoso servidor de pesquisas corporativas baseada no

11 http://lucene.apache.org/solr/

projeto *Apache Lucene*[12] e o *Carrot2*[13], um *framework* de recuperação, agrupamento e visualização de documentos e conteúdos web.

De forma geral, como apresentado na Figura 16, para utilização da AVS, um coletor de dados indexa constantemente todos os relatórios de erros provindos da base principal (Coleta dos dados). Com isso, o processo de buscas e identificação de relatórios de erros se divide em 3 etapas: sempre que um erro é encontrado, o usuário (*e.g.*, testador, usuário final, etc.) fornece ao sistema uma breve descrição inicial da situação de erro através de uma consulta (Entrada); Em seguida, o sistema executa automaticamente três fases, descritas abaixo (Processamento); e, no final, através de uma análise manual, o usuário decide se deve ou não criar um relatório de erro com base na lista de resultados recuperadas pelo sistema (Análise dos resultados).

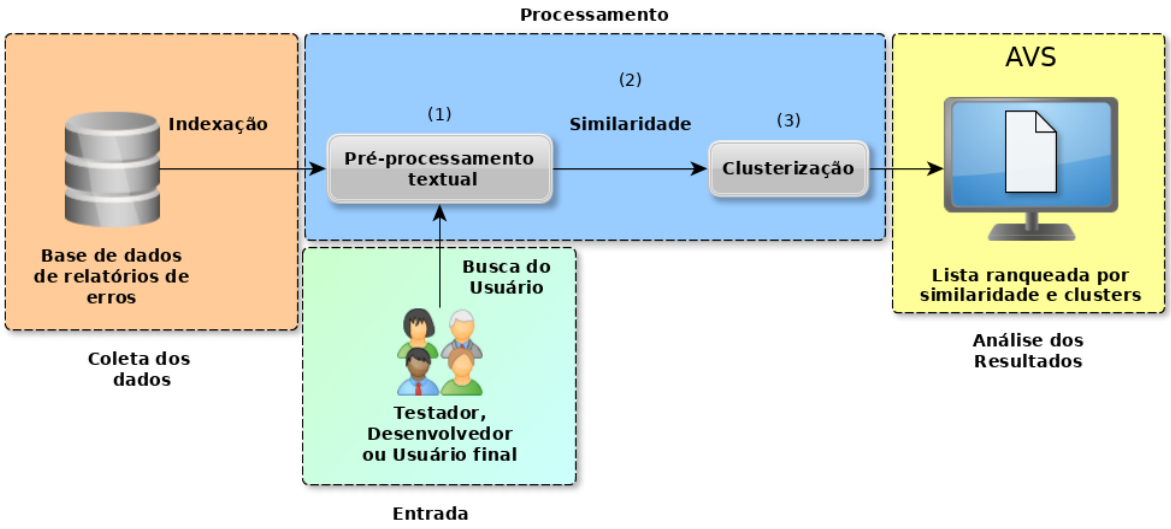

Figura 16 – Arquitetura da AVS - Fonte: Autor

As 3 fases da etapa de Processamento são descritas da seguinte forma:

Na fase **(1)**, o passo de Pré-processamento textual recebe como entrada a base de dados que é indexada preliminarmente (*i.e.*, relatórios criados anteriormente) e a descrição fornecida pelo testador (*i.e.*, consulta). Com as informações obtidas, vários recursos de processamento de texto ajustam tanto a pesquisa quando o conteúdo da base para um melhor entendimento e trabalho pelo sistema.

12 http://lucene.apache.org/

13 https://project.carrot2.org/

Na fase **(2)**, utilizando uma medida probabilística de similaridade baseada na função *Okapi BM25*[14], uma lista de relatórios de erros ordenados é retornada. Esta ordem é feita de acordo com a semelhança dos resultados com a consulta informada pelo usuário.

Na fase **(3)**, por fim, após receber a lista com os resultados obtidos do passo anterior, é realizado um processo de agrupamento (*i.e.*, *clustering*) dos relatórios. Com isso, é criada também uma lista de rótulos ordenados por relevância, onde os documentos são agrupados e divididos de acordo com suas semelhanças às descrições de cada rótulo.

Nas próximas subseções serão explanados maiores detalhes de cada uma destas fases.

4.3 INDEXAÇÃO DA BASE DE RELATÓRIOS DE ERROS

Uma vez que os relatórios de erros são documentos compostos por diversos campos, na coleta de dados da AVS são estabelecidas regras para indexar e armazenar apenas os pontos com conteúdos considerados relevantes. Para isso, foram selecionados os principais campos que influenciam diretamente em uma busca. Estes campos foram: *Summary* e *Description* (por armazenarem o resumo e toda descrição detalhada do erro), *Status, Resolution*, *Date created, Date updated, Type* e *Key* (para serem capazes de prover filtros nas pesquisas).

Outro fator importante para a escolha desses campos foi a necessidade de dispor de uma apresentação mais simples e amigável de cada um dos relatórios fornecidos na lista de resultados ao usuário final. Dessa forma, apenas os conteúdos considerados essenciais para a análise seriam exibidos nos resultados.

4.4 PRÉ-PROCESSAMENTO DE DADOS NA AVS

O pré-processamento da AVS foi criado para coletar e tratar informações relevantes de dentro dos relatórios armazenados na base de dados e nas buscas geradas pelos usuários, levando em consideração aspectos relacionados a erros de softwares. Apesar do domínio estudado neste trabalho ter sido focado em um projeto mobile utilizando o sistema *Android*[15], esta proposta visa ser aplicável genericamente em qualquer domínio. Para isso ser possível, é necessário ressaltar a importância de análises e estudos para entender as principais conjunturas do domínio alvo, onde de acordo com os tipos de dados são necessárias

14 https://lucene.apache.org/core/7_0_1/core/org/apache/lucene/ search/similari ties/BM25Similarity.html

15 https://www.android.com/

adaptações às necessidades de tratamento do conteúdo para com isso prover um melhor aproveitamento de informações. Ou seja, de acordo com a linguagem ou tipo de documento a ser manipulado, alguns dos processos podem precisar de pequenas modificações.

Ao final dos estudos no domínio especificado, foi definido um fluxo de pré-processamento, representado pela Figura 17:

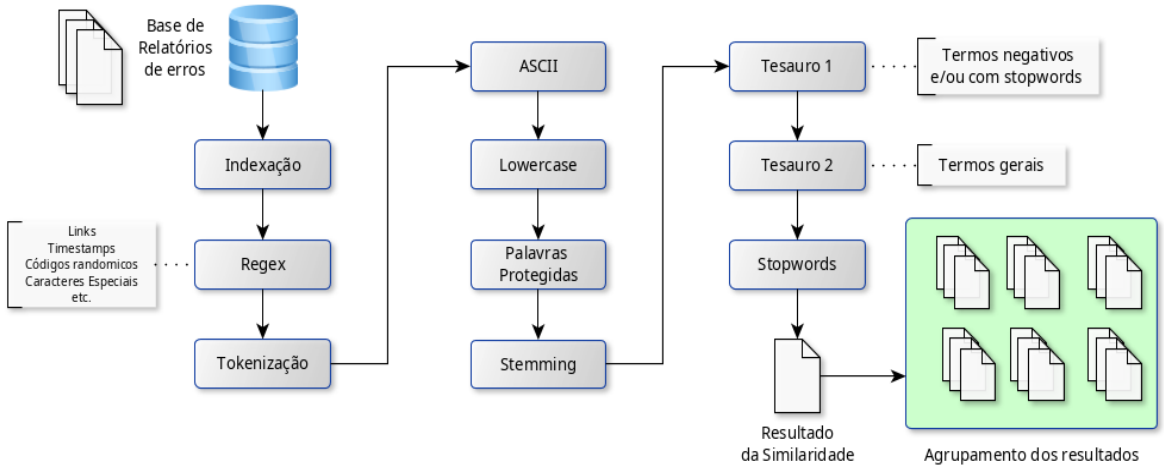

Figura 17 – Fluxograma de pré-processamento na AVS - Fonte: Autor

Nas próximas subseções serão abordadas cada uma das etapas relacionadas ao fluxo de pré-processamento dos dados pela AVS utilizadas para o domínio mobile.

4.4.1 Remoção de expressões irrelevantes

Com o intuito de realizar pesquisas por conteúdos mais relevantes, na AVS foram criadas regras baseadas em expressões regulares (*i.e.*, *regex*) para remover itens irrelevantes que um usuário pode imputar em sua busca. Entre os itens irrelevantes podem ser citados:

- *Timestamps* (*i.e.*, tempo exato de ocorrência de um erro, datas, etc);
- UIDs (*e.g.*, *hashes* e *ids* randômicos);
- Símbolos e caracteres acentuados (levando em consideração a língua inglesa);
- entre outros.

O propósito deste processo é remover todos os itens considerados irrelevantes e que não podem ser adicionados ao documento de *stopwords* devido sua mutabilidade.

4.4.2 Tokenização e Padronização de termos

Na AVS o processo de tokenização é a base para a padronização inicial dos termos. Neste processo são aplicadas a remoção de caracteres especiais e acentuações, e a transformação da sensibilidade de letras maiúsculas e minúsculas (*i.e.*, *case-sensitive*).

4.4.3 Stemmização e Palavras Protegidas

O recurso de stemização tem por objetivo normalizar palavras de forma a evitar discrepâncias entre termos diferenciados pelas suas conjugações. Com isso é possível aumentar a cobertura dos resultados das buscas. Na AVS, esse processo contribui também para diminuir a expansão excessiva de termos dos tesauros (*i.e*, a quantidade de palavras adicionadas). Para isso, como *stemmer* padrão, a AVS utiliza o *Porter Stemmer*.

Todavia, para resolver problemas relacionados a "*agressividade"* (*i.e.*, o nível de mudança dos termos) do algoritmo de stemização*,* a AVS proporciona um filtro no qual palavras que podem perder ou mudar seu contexto após stemizadas*,* são "protegidas" do processo de stemização. Dessa forma, ao ser realizada uma busca, as palavras que podem perder contexto ao serem stemizadas, não são alteradas, evitando assim a perda de relevância. Para esclarecer situações onde a stemização pode prejudicar a relevância dos resultados, alguns exemplos são mostrados a seguir:

- **Mistura de contextos**:
    - ***us*** (*i.e.*, *United States*), ***useful*** (*i.e.*, útil)*,* ***use*** (*i.e.*, uso), após stemizadas todas são reduzidas à ***us***;
    - ***display*** (*i.e.*, no contexto mobile pode se referenciar a tela, *screen*, etc.), ***displays*** (*i.e.*, mostra), ***displayed*** (*i.e.,* mostrou), ***displaying*** (*i.e.*, mostrando), após stemizadas todas são reduzidas à ***displai***;
- **Alteração de siglas**:
    - ***gps, sms***, ***fps, us***, que após a realização da stemização perdem o ***s***. Na situação de ***us*** após a stemização torna-se uma *stopword* (*i.e.*, resta apenas a letra **u**) sendo considerada irrelevante. Quanto aos demais casos, a remoção do ***s*** modifica as siglas, o que por sua vez pode prejudicar na identificação dos termos no processo de tesauro*;*

- **Alteração de substantivos próprios**
    - Outro caso observado está relacionado ao uso de substantivos próprios (*i.e.*, nomes de pessoas ou objetos[16]) terminados em ***s***, ***y***, ***e*** e ***es***, que após a stemização são alteradas de forma a modificar sua estrutura (*i.e.*, o ***e*** e o ***es*** são removidos e o ***y*** é transformado em ***i***).

Com a proteção dos termos ligados as situações descritas anteriormente, é possível criar relações corretas entre contextos e evitar que os resultados das buscas percam precisão devido as possíveis alterações.

4.4.4 Tesauro

No domínio de sistemas *mobile*, muitos são os termos considerados equivalentes entre si. Além disso, em inúmeras situações, jargões técnicos e neologismos são comumente utilizados para expressar componentes de um software ou ações e eventos ocorridos durante o seu uso. Nestes casos, o tesauro tem papel essencial como meio de agrupamento de informações relacionadas entre si, capaz de armazenar de forma simples uma base de conhecimento, que ao ser utilizado em sistemas de busca, proporciona um melhor entendimento entre expressões de um determinado contexto.

Contudo, devido as particularidades que um domínio pode ter, é necessário todo um processo envolvendo trabalho manual para identificação e estruturação dos termos (*i.e.*, com a ajuda de especialistas do domínio, estudo sobre componentes e características dos produtos e pesquisas na base de dados), visto que ainda não existem ferramentas que possam contribuir para a identificação de termos específicos à um domínio de forma automática.

Dessa forma, para construção de um tesauro adequado ao domínio em questão, seis cenários principais foram elencados e levados em consideração:

1. **Flexões e derivações de termos:**

Inicialmente, visando o controle de expansão a longo prazo do tesauro (*i.e.*, a escalabilidade do tamanho do seu conteúdo), foi implementada uma estratégia na qual o processo de stemização dos termos é feito em um passo anterior ao tesauro. Assim, é possível contribuir para que em situações envolvendo flexões e derivações dos termos, não seja

16 No domínio estudado, foram encontrados vários nomes de próprios (ex. apelidos, nomes de pessoas, etc.) relacionados aos produtos.

necessário colocá-las todas dentro do tesauro. Para isso ser possível, todos os termos incluídos no tesauro são anteriormente stemizados (*i.e.*, o tesauro é construído com radicais dos termos), gerando um fluxo de trabalho similar ao representado na Figura 18.

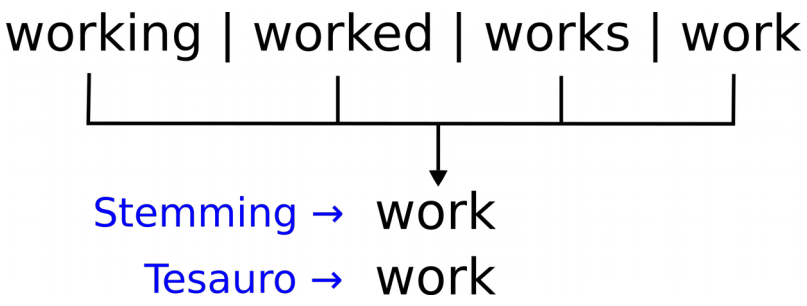

Figura 18 – Stemização de termos e tesauro stemizado - Fonte: Autor

onde, o usuário entra com o termo em qualquer uma de suas flexões ou derivações, em seguida a stemização normaliza o termo para seu radical, e esse radical é enfim comparado ao radical contido dentro do tesauro. Com o tesauro stemizado, foi possível diminuir seu tamanho e assim trabalhar com termos múltiplos de uma forma mais simplificada.

2. **Termos simples considerados equivalentes à múltiplos termos (e vice-versa)**:

Cenário onde há equivalência entre múltiplos termos e termos simples. Em casos como esses, o objetivo do tesauro é criar uma relação entre essas situações aumentando a cobertura das pesquisas às possíveis variações de termos. Nestas circunstâncias, é comum também serem associadas situações de erros entre múltiplos termos.

Google Play Store	→	*Play Store*
Obsolete	→	*Out of date*
Whats app	→	*Whatsapp (espaço acrescentado no termo)*
Homescreen	→	*Home Screen (termos sem espaço)*

3. **Termos expressados ou descritos de forma incorreta**:

Cenário onde os usuários por falta de conhecimento ou por descuido, digitam termos de forma incorreta durante a construção de uma pesquisa, e consequentemente acabam criando pesquisas com perda de informações. Nessas situações, o uso do tesauro pode funcionar como um "*corretor*" para termos digitados de forma errada interpretando e ligando situações incorretas as suas respectivas correções.

Forma errada		Forma correta
LinkedIn	→	*LinkedIn*: (*i* trocado por *l*)
*Instagra**n***	→	*Instagram*: (*m* trocado por *n*)

4. **Uso de siglas e números**:

Cenário onde os usuários utilizam termos simplificados (*i.e.*, siglas, abreviações, reduções de termos etc.) para substituir termos longos ou conjuntos de múltiplos termos e números representados de forma numérica ou textual. Em situações como esta, o tesauro facilita a ligação entre números e termos simples com seus equivalentes na forma por extenso.

Sigla		Significado
FB	→	*Facebook*
BT	→	*Bluetooth*
PIP	→	*Picture in Picture*
FPS	→	*Finger Print Sensor*
1	→	*One*
9	→	*Nine*

5. **Verbos irregulares**:

Outro cenário em destaque, está no uso de verbos irregulares, que são termos que sofrem alterações em seus radicais ou em suas desinências, diferenciando-se do modelo de termo a quem pertencem. Para situações como essa a stemização não é suficiente para considerar a semelhança entre os radicais dos termos. Para resolver este caso, o tesauro funciona como meio de indicação de equivalência dos termos, onde para cada conjunto de verbos é definido qual termo será representado como principal para o conjunto dos demais.

Verbos irregulares		Termo principal
begin, began, begun	→	*begin*
draw, drew, drawn	→	*draw*
forget, forgot, forgotten	→	*forget*

freez, froze, frozen → *freez* ***(Freeze)***

6. **Funcionalidades ou expressões contendo *stopwords*:**

Por fim, são vislumbradas ocasiões onde termos comumente considerados irrelevantes (*i.e.*, *stopwords*) tornam-se cruciais para a interpretação completa de um contexto (*e.g.*, funcionalidades, estados, etc.), o que por sua vez demanda um tratamento adequado para evitar perda de informações relevantes. Exemplos:

Do not Disturb (***do*** e ***not***)

Out of Memory (***out*** e ***of***)

Picture in Picture (***in***)

it_it (representação da língua italiana)

Three fingers to screenshot (***three*** e ***to***)

Um exemplo de destaque da importância do tesauro está na situação de erro chamada "*Application not work*", comumente conhecida pela sigla ANR. Nesta sentença além de diversas flexões para o verbo, há uma *stopword* considerada relevante para interpretação do contexto. A partir do uso de linguagem natural, está expressão pode assumir inúmeras formas, incluindo desde variações feitas com a mudança dos termos "*Application*" para "*App"* e de todas as flexões da palavra "*work*", até a maioria das formas negativas substitutas ao termo "*not*". Para melhor visualização destas possibilidades, a Figura 19 apresenta algumas das possíveis formas de descrição da expressão *ANR* dentro de um domínio de sistemas mobile.

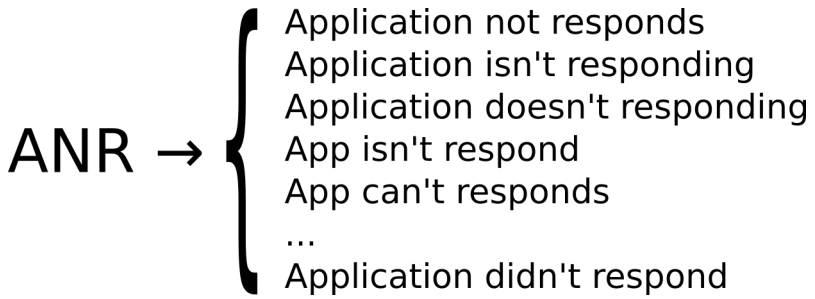

Figura 19 – Exemplos de variações da sigla ANR - Fonte: Autor

Como são muitos os cenários onde o tesauro se aplica (*e.g.*, neologismos, jargões, siglas, sinônimos, termos equivalentes, etc.), a realização de uma filtragem de todas as situações descritas anteriormente foi fundamental para adequar o tesauro ao domínio. Apesar da necessidade de trabalho manual para sua construção, o propósito do tesauro é proporcionar um nível de cobertura maior e mais adequado às informações e com isso aumentar a precisão dos resultados das buscas.

4.4.4.1 Múltiplos Tesauros

Antecipando uma solução para problemas relacionados a escalabilidade do tesauro mediante as variações excessivas de termos (*e.g.*, na situação de ANR), foi necessário criar uma estratégia capaz de diminuir a quantidade de itens de dentro do tesauro a longo prazo.

Para isso, foram criados 2 tesauros: um contendo todos os termos de negação e outro contendo termos gerais (*i.e.*, sinônimos, neologismos, siglas, etc.).

A estratégia aplicada levou em consideração o tipo do domínio. Como o trabalho de submissão de relatórios de erros em sua grande maioria está ligado a situações negativas (*e.g.*, erros, defeitos, falhas, etc.), ou seja, um usuário reporta algo negativo encontrado ao utilizado ou testar um software, todas os termos de negação são comumente utilizados para descrever um problema (*e.g.*, ***not*** *works*, ***isn't*** *open*, etc.). Ao agrupar todos os termos de negação do inglês (*e.g.*, *not*, *cannot*, *does not*, *isn't, didn't*, etc.) em uma única situação, seria possível diminuir as variações de expressões de múltiplos termos que continham *stopwords* de negação dentro do tesauro. Exemplo:

not → no, don't, ..., didn't, did not

Dessa forma, foi possível diminuir a quantidade de variações para expressões que continham *stopwords* negativas. Com isso, no tesauro de termos gerais, a representação de ANR ficou da seguinte forma:

anr → not respond, app not respond, …, applic not respond

Ou seja, para qualquer uma das variações, o pré-processamento irá transformá-las na sigla "anr". Desta forma, como apresentado na Figura 20, com a utilização do processo de stemização e dos 2 tesauros combinados, é possível obter um nível total de cobertura do termo sem necessitar de um tesauro extenso.

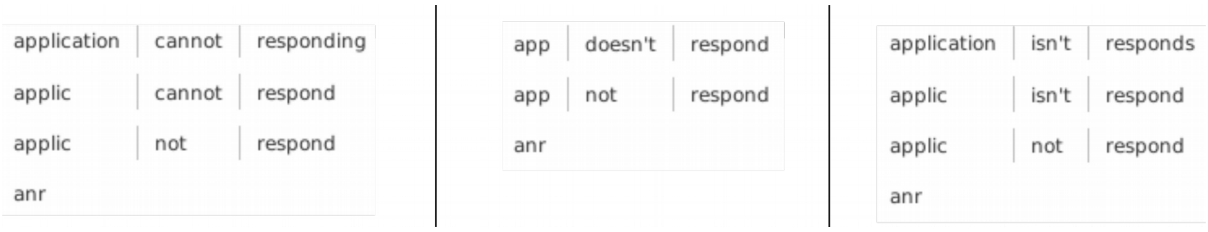

Figura 20 – Exemplo de pré-processamento do termo ANR - Fonte: Autor

4.4.5 Remoção de Stopwords

A remoção de termos irrelevantes foi incluída como última etapa justificando as situações explicadas anteriormente onde *stopwords* podem ser consideradas relevantes a um determinado contexto, como também foi explicado na subseção 3.4.3. Com isso, todas as *stopwords* remanescentes são removidas e assim o papel de remoção de termos irrelevantes é concluído, diminuindo a dimensionalidade do conteúdo e melhorando a relevância dos dados.

4.5 BUSCA COM SIMILARIDADE

Como já mencionado, as principais ferramentas de gerenciamento de relatórios de erros do mercado contêm sistemas de buscas baseados no modelo booleano, que mesmo provendo simplicidade e resultados precisos, porém não ordenados, ainda são os mais conhecidos e utilizados por empresas de software.

Em consequência disso, a AVS foi estruturada para prover um sistema de buscas com resultados baseados em similaridade textual utilizando o algoritmo BM25. A escolha do algoritmo se deu ao avaliar que no domínio em questão, os relatórios de erros comumente têm tamanhos diferentes (*i.e.*, existem relatórios muito curtos e outros muito longos), e isso favorece a utilização do modelo probabilístico, já que o mesmo se adapta muito bem em buscas onde há tamanhos variados de documentos em relação a consultas pequenas (TIAN; LO; SUN, 2012) independente do domínio utilizado. No entanto, apesar do recurso principal de buscas ser baseado em similaridade textual, a AVS também é capaz de conciliar seu sistema com buscas booleanas, possibilitando que o usuário crie buscas de acordo com sua experiência. Neste caso em particular, o usuário terá a possibilidade de utilizar os operadores booleanos (*e.g.*, *AND*, *NOT* e *OR*), junto aos recursos oferecidos pela similaridade de termos. Exemplo:

- (*Camera not working*) **AND** (*The application crashes*). Nesta situação o usuário terá os recursos de similaridade para os termos entre parênteses, além da regra booleana *AND* para definição dos resultados. Assim, o sistema irá buscar tudo o que for similar as sentenças "*Camera not working"* e "*The application crashes",* porém, com o uso do operador *AND*, só aparecerão resultados onde as semelhanças entre uma situação e a outra juntas forem encontradas.

Outro fator importante com relação ao sistema de buscas, está na possibilidade de definição de filtros de acordo com o campo do relatório de erro (*e.g.*, *summary*, *description*, *status*, etc.), o que proporciona ao usuário recursos para buscas específicas. Além disso, é possível também, a utilização de expressões regulares na construção das consultas. Com essas possibilidades, a AVS é capaz de oferecer diversas alternativas para o usuário construir suas pesquisas de acordo com suas necessidades. Essas características de filtragem de campos, junção de busca booleana com similaridade textual e consultas com expressões regulares, são herdadas do mecanismo de buscas oferecido pelo *Apache Solr*.

Ao final, o objetivo deste processo é exibir de forma decrescente, pelo percentual de semelhança, uma lista ordenada de relatórios de erros relacionados à pesquisa elaborada pelo usuário.

4.6 AGRUPAMENTO DOS RESULTADOS

Como última fase da etapa de processamento, o agrupamento de dados tem como propósito, fornecer uma visão adicional de possíveis correlações entre relatórios provenientes do resultado obtido da similaridade no processo de buscas. O objetivo desta fase é oferecer a possibilidade de descoberta de conhecimento sobre os dados agrupados.

Como citado anteriormente na subseção 4.2, para esta finalidade, a AVS conta com o uso do *framework* Carrot2, onde sua aplicação web[17] foi adaptada para servir como interface visual do usuário. Através do Carrot2, foi possível fornecer algoritmos capazes de agrupar as informações semelhantes entre si dos resultados de cada busca. O Carrot2 proporciona ao usuário 3 algoritmos para agrupamento das informações, o *Lingo*, o *STC* e o *K-Means.* Com eles, o usuário pode escolher qual algoritmo ele utilizará para o agrupamento e a partir disso avaliar melhor os resultados obtidos.

17 https://project.carrot2.org/download-webapp.html

Ao final desse processo, o usuário recebe além da lista com os resultados da similaridade, um conjunto de rótulos que contribuirá também para a descoberta de novas áreas e duplicações entre os relatórios provenientes de sua consulta.

4.7 INTERFACE VISUAL DA AVS

Nesta subseção será mostrado o funcionamento e a interface principal da AVS. Como apresentado na Figura 21, os principais itens da AVS são:

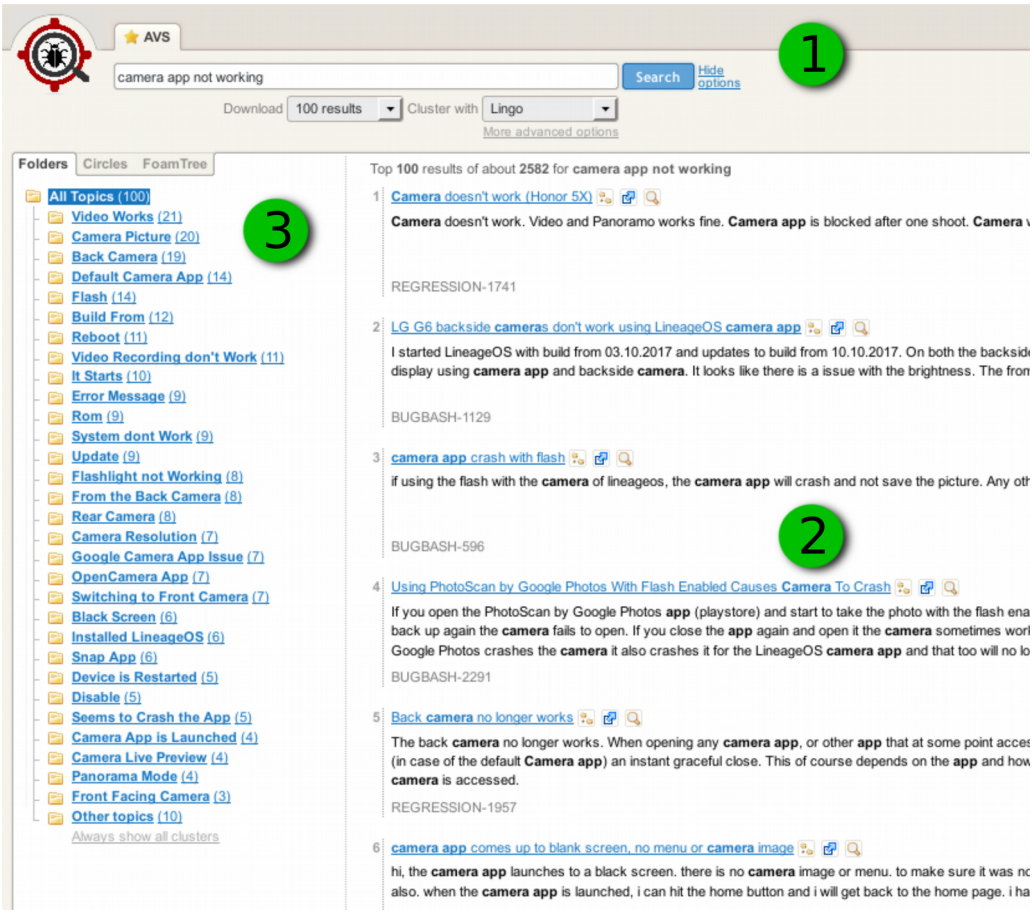

Figura 21 – Interface visual da AVS - Fonte: Autor

1.

Pesquisa:

Nesta área encontram-se as principais funcionalidades para a realização da busca do usuário. Essa parte é responsável também, pela definição de filtros de pesquisa. Além de repassar a consulta, o usuário também pode definir, o número de resultados exibidos e o tipo de algoritmo para agrupamento dos resultados.

2. **Resultados da busca**:

Essa parte é responsável por exibir de forma ordenada todos os relatórios relacionados à consulta realizada pelo usuário. Cada relatório é apresentado com seu *summary* como título, um trecho do conteúdo de sua *description*, além de sua *key*. Além disso o Carrot2 oferece também uma pré-visualização do relatório (*i.e.*, o relatório é visualizado em uma cortina) através de um botão representado pelo ícone de uma lupa.

3. **Agrupamento dos resultados**:

Nesta área são exibidos os agrupamentos de acordo com os resultados das buscas. Graças ao uso do Carrot2, a visualização dos agrupamento pode ser feita de 3 formas, através de uma árvore de pastas (*i.e.*, *folders*) representado na Figura 22, gráfico circular (*i.e.*, *circles*), representado na Figura 23 e através de uma árvore de espuma (*i.e.*, *foamtree*) representado na Figura 24.

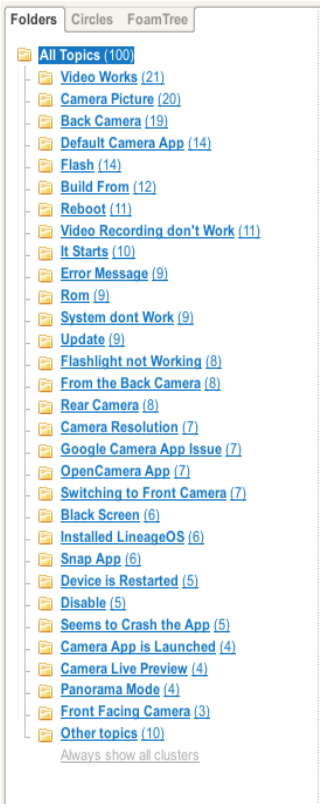

Figura 22 – Agrupamento através de pastas - Fonte: Carrot2 webapp

Figura 23 – Agrupamento com gráfico circular - Fonte: Carrot2 webapp

Figura 24 – Agrupamento com árvore de espuma - Fonte: Carrot2 webapp

4.8 CONSIDERAÇÕES FINAIS

Neste capítulo, foi apresentada uma ferramenta de buscas e análise de relatórios de erros, criada com o propósito de mitigar a criação de relatórios duplicados de forma mais ágil e precisa. Assim, sua arquitetura foi descrita, com suas fases e etapas, além dos principais recursos de melhoramento empregados para o pré-processamento, pesquisa e análise de relatórios de erros. Apesar da ferramenta ter sido construída com base nas necessidade de um domínio específico, sua arquitetura pode servir como base para implementação entre outros domínios, dada a sua flexibilidade de mudanças e adaptabilidade a novos recursos.

O próximo capítulo apresenta um estudo de caso realizado dentro de um projeto privado da Motorola Mobility[18] (*a Lenovo Company*). Neste estudo, a AVS foi utilizada e avaliada em paralelo com uma ferramenta de referência da linha de produção de testes do projeto em questão.

18 https://www.motorola.com.br/

5

ESTUDO DE CASO

Neste capítulo, será descrita a análise de avaliação da AVS, assim como os resultados obtidos. Vale ressaltar que o estudo de caso foi conduzido e realizado em um ambiente de produção da Motorola (*a lenovo company*), onde o AVS foi testado em um ciclo real de testes. Na empresa em questão, o Jira é o gerenciador de relatórios de erros padrão. A princípio, na subseção 5.1 será descrita a metodologia empregada no estudo de caso, na subseção 5.2, o processo de criação do corpus utilizado durante os testes, na subseção 5.3, o processo de seleção dos participantes, em seguida, na subseção 5.4 serão detalhados os resultados de todo o procedimento, e ao final, a subseção 5.5 que é destinado a discutir as lições aprendidas.

5.1 ANÁLISE DAS BUSCAS

A avaliação das buscas foi baseada nas métricas de Precisão, Cobertura e Medida F como calculadas por (CREMONESI; KOREN; TURRIN, 2010). Resumidamente, cada participante teve acesso a um vídeo ou imagem relacionado a um erro, para através disso definir uma consulta textual como entrada para a AVS ou Jira.

Como exemplo, considere um relatório (**x**) no estudo de caso, foi verificado se **x** era encontrado entre os resultados de uma pesquisa quando o usuário, utilizando consulta textual como entrada, procura por sua ocorrência na base de dados da ferramenta. Vale salientar, que em um sistema de recuperação preciso, é esperado que **x** seja apresentado entre as posições superiores do ranking de resultados.

Para se obter os resultados de precisão, cobertura e medida F dos relatórios encontrados em cada uma das buscas, foram utilizadas as seguintes fórmulas:

$$precisão = \frac{hit}{N} \times 100 \tag{9}$$

$$cobertura = \frac{hit}{T} \times 100 \quad (10)$$

$$F = \left(2 \times \frac{precisão \times cobertura}{precisão + cobertura} \right) \times 100 \quad (11)$$

Onde, na precisão, para cada relatório corretamente (**x**) encontrado pelo usuário, um *hit* foi atribuído (*i.e.*, *hit* é igual a *1* caso o relatório seja encontrado ou *0* para não encontrado) e sua posição *N* (*i.e.*, a posição onde ele foi encontrado no ranking) foi registrada. Por exemplo, no caso de um relatório corretamente encontrado na terceira posição (*i.e.*, *N* = 3) do ranking, sua precisão seria calculada da seguinte forma: (1/3) * 100 = ~ 33%.

Para a cobertura, da mesma forma como na precisão, além do *hit* atribuído (*i.e.*, *1* para encontrado e *0* para não encontrado), foi atribuída a quantidade *T*, que representa o número total de resultados que deveriam ser encontrados (*i.e.*, no caso deste estudo de caso para cada busca sempre haveria 1 relatório a ser encontrado, ou seja, *T* = 1). Por exemplo, para cada relatório corretamente encontrado era calculado (1/1) * 100 = 100%, caso contrário (0/1) * 100 = 0%.

O intuito deste estudo de caso foi replicar de forma controlada um ambiente semelhante ao de produção de testes. Como cada busca deveria ser feita de forma independente, ou seja, um relatório por vez, a avaliação por *hits* se mostrou fundamental para interpretação dos resultados, visto que assim, seria possível simular um ambiente real onde o usuário buscaria por cada erro de forma individual.

Para realização das buscas do estudo de caso, foi criado um corpus e selecionados participantes seguindo os critérios explicados nas próximas subseções.

5.2 CRIAÇÃO DO CORPUS DO ESTUDO DE CASO

Com o objetivo de realizar a avaliação, foi criado um corpus contendo cerca de 750 mil relatórios obtidos de uma base através da plataforma *Google Cloud*[19]. Cada relatório do corpus era composto pelos campos:

- ***summary*** (*i.e.*, resumo);
- ***description*** (*i.e.*, descrição);

19 https://cloud.google.com

- ***id*** (*i.e.*, chave identificadora);
- ***status*** (*i.e.*, situação do relatório);
- ***resolution*** (*i.e.*, resolução);
- ***date created*** (*i.e.*, data de criação);
- ***date updated*** (*i.e.*, data de atualização).

Do corpus construído, para o estudo de caso foi criado uma simulação de um ambiente de testes, onde buscas deveriam ser realizadas, foram selecionados 10 relatórios de forma aleatória, os quais, obedeciam aos seguintes critérios:

a) Os relatórios deveriam ter suas datas de criação em um intervalo dentro dos 5 últimos meses;

b) Cada relatório deveria conter ao menos 1 arquivo (*e.g.*, vídeo ou imagem) que descrevesse o erro de forma clara. O objetivo desses arquivos seria fornecer aos participantes do estudo de caso detalhes sobre o erro reportado, capazes de servir como guia para realização de buscas;

c) O relatório não poderia estar relacionado (*i.e.*, ter sido criado ou revisado) ou anteriormente já ter sido verificado por nenhum dos participantes envolvidos no estudo de caso. Para garantia deste critério foram selecionados relatórios criados por times fora do projeto em questão (*i.e.*, de outras unidades da empresa).

5.3 SELEÇÃO DO GRUPO DE PARTICIPANTES

Para composição do grupo de participantes desse estudo de caso, foram selecionadas de forma aleatória, 6 pessoas sem nenhuma experiência tanto com testes quanto no uso de GRE (*i.e.*, novatos dentro do projeto). A escolha de participantes sem experiência foi necessária para avaliar também a facilidade de uso para novos usuários.

Essas pessoas foram submetidas a um treinamento prévio para que fossem capazes de conhecer as funcionalidades básicas oferecidas tanto pela AVS quanto pelo Jira (*e.g.*, explicação dos recursos básicos e da representação dos resultados). Por fim, cada usuário teve um tempo de 5 minutos por arquivo de relatório apresentado (*i.e.*, imagem ou vídeo do erro), para analisar cada uma das situações e fazer anotações que julgasse necessárias para o guiar na realização das pesquisas.

Para criação das consultas, foi recomendado a utilização de resumos contendo palavras chaves que descrevessem de forma breve e sucinta a situação de erro, entretanto, o participante tinha a liberdade de criar sua consulta a partir de seus próprios conhecimentos.

Cada avaliação foi feita de forma individual, ou seja, nenhum dos participantes interagia com os demais durante o processo. Inicialmente as buscas foram realizadas no Jira e em seguida na AVS. Todas as buscas foram realizadas de acordo com os seguintes critérios:

- O usuário poderia criar uma única consulta textual no Jira (seguindo ou não as recomendações fornecidas durante o treinamento) e esta mesma consulta seria reutilizada na AVS.
- Cada usuário poderia realizar as 10 buscas de forma aleatória, isto é, sem seguir uma sequência pré-definida de buscas.
- Assim que encontrado o relatório alvo, o participante tinha que alertar e indicar a posição onde o relatório havia sido encontrado, para a partir daí ser analisado e confirmado o *hit*.

5.4 RESULTADOS DA ANÁLISE

Na Tabela 1, são apresentados os resultados de cada *hit* obtidos entre os usuários utilizando a AVS.

Tabela 1 – Resultados das buscas por posição utilizando a AVS – Fonte: Autor

Relatório	Usuário 1	Usuário 2	Usuário 3	Usuário 4	Usuário 5	Usuário 6
01	2	2	3	1	3	1
02	13	11	14	3	7	0
03	9	9	7	2	2	6
04	4	2	2	10	2	2
05	6	1	11	4	2	2
06	4	4	4	3	3	1
07	3	4	2	1	1	1
08	1	1	2	1	1	3
09	1	1	5	1	1	1
10	1	1	11	1	1	1

Como pode ser observado dentre os resultados, apenas em uma das buscas (realizada pelo usuário 6) não foi encontrado um dos relatórios do estudo de caso[20]. Este fato ocorreu porque o usuário interpretou o erro de uma forma diferente dos demais e essa interpretação o fez pesquisar por um contexto diferente do representado pelo relatório. Apesar disso, todas as buscas realizadas apresentaram bons resultados, encontrando os relatórios em posições bem próximas ao topo do ranking.

Na Tabela 2, são apresentados os resultados de cada *hit* obtidos entre os usuários utilizando o Jira.

Tabela 2 – Resultados das buscas por posição no Jira – Fonte: Autor

Relatório	Usuário 1	Usuário 2	Usuário 3	Usuário 4	Usuário 5	Usuário 6
01	0	3	0	0	8	1
02	0	0	15	0	70	0
03	14	0	0	0	2	0
04	0	2	0	0	4	0
05	0	0	0	4	9	4
06	0	0	0	1	1	20
07	0	0	0	1	3	1
08	0	1	4	1	1	4
09	42	0	0	1	1	1
10	1	1	0	6	1	1

Como citado anteriormente, devido as limitações do Jira e a necessidade de conhecimentos específicos a respeito de buscas booleanas (*i.e.*, baseadas em SQL), em grande parte das buscas os relatórios não foram encontrados, e em casos mais específicos, vieram em posições muito distantes do topo.

Nas Tabelas 3 e 4, são apresentados os resultados gerais das médias de precisão, cobertura e medida F realizados tanto com a AVS quanto como o Jira.

20 Foram consideradas como não encontrados os relatórios onde o usuário desistia de buscá-lo, o que por sua vez, em um ambiente real poderia ser interpretado como se o erro buscado em questão não houvesse sido reportado anteriormente.

Tabela 3 – Médias dos resultados utilizando a AVS – Fonte: Autor

Precisão	53.99%
Cobertura	98.33%
Medida F	62.73%

Tabela 4 – Médias dos resultados utilizando o Jira – Fonte: Autor

Precisão	30.91%
Cobertura	53.33%
Medida F	38.34%

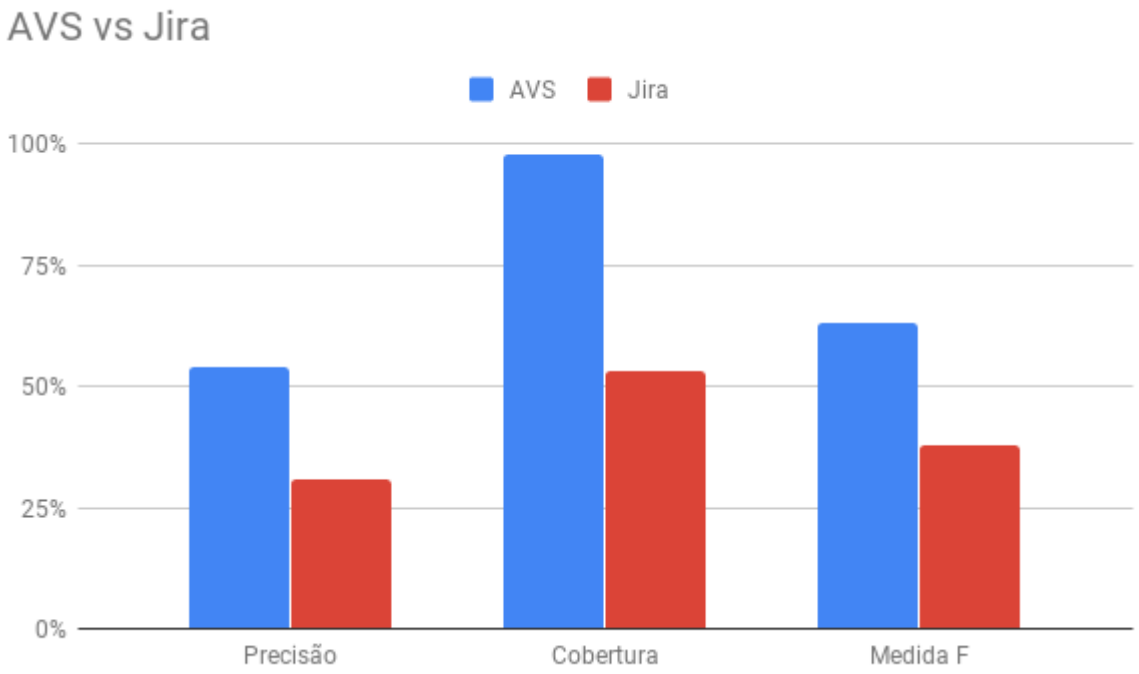

Figura 25 – Médias de Precisão, Cobertura e Medida F - Fonte: Autor

Como apresentado na Figura 25, a AVS alcançou ganhos em precisão de cerca de **23%**, cobertura de **45%** e medida F de **24%**, demonstrando um resultado satisfatório na melhoria do processo de busca por relatórios existentes na base de relatórios de erros criada.

5.5 LIÇÕES APRENDIDAS

Durante o estudo de caso, alguns pontos foram analisados para verificar as principais dificuldades nas buscas e se a implementação dos objetivos específicos obteve sucesso. Para

isso, foram estudadas duas situações, a forma como os usuários realizavam as buscas e a utilidade das funcionalidades oferecidas pela AVS em relação ao Jira.

5.5.1 Buscas dos usuários

Inicialmente, foi observado que os conhecimentos sobre termos técnicos, nomes de componentes de um *smartphone,* além do nível de experiência sobre a língua inglesa, tiveram influência significativa na realização das buscas. Isso se deu por que além dos conhecimentos específicos, o inglês era o idioma principal nos relatórios.

Entre os usuários com nível básico de inglês ou que não tinham experiência técnica sobre um *smartphone*, foram apresentadas indecisões na construção das frases de consulta, isto consequentemente, fez com que em várias ocasiões os relatórios buscados aparecessem em posições mais distantes do topo ou não fossem encontrados (principalmente na utilização do Jira). Além disso, também foi constatada a necessidade de mais tempo durante as pesquisas utilizando o Jira. Isso aconteceu porque o sistema de buscas oferecido pelo Jira se tornou o principal fator de complexidade, devido à falta de tratamento prévio do texto e a ausência de um mecanismo de similaridade nos resultados das pesquisas, fazendo assim, com que os usuários necessitassem realizar buscas mais complexas para tentar encontrar os relatórios.

5.5.2 Funcionalidades da AVS

Com os relatórios indexados, o acesso às respostas para as buscas se mostrou muito mais ágil, sendo entregue visualmente de forma instantânea. Essa característica contribuiu tanto na agilidade do processo quanto na diminuição da carga de uso da base de dados, visto que para este acesso, a AVS trabalhava de forma independente do Jira com seu próprio armazenamento indexado dos dados.

Além disso, com o uso do tratamento de texto, dos tesauros e da similaridade, foi averiguado o aumento da cobertura e da precisão das respostas, uma vez que os relatórios buscados eram encontrados em posições mais próximas ao topo em comparação ao Jira. Outro fator, foi a facilidade de utilização e acesso às informações apresentadas, que indicou um melhor aproveitamento das pesquisas, devido à simplicidade oferecida tanto no momento de construção das sentenças de buscas (*i.e.*, *querys* de buscas) quanto nas respostas (*i.e.*, somente os principais campos de um relatório para análise dos resultados eram exibidos).

Os resultados destas situações se mostraram favoráveis diante de um cenário onde a complexidade proporcionada pelas buscas boolenas, a sobrecarga causada por buscas extensas e a diversidade da linguagem natural, são os principais causadores de dificuldades para se encontrar um relatório criado anteriormente.

5.6 CONSIDERAÇÕES FINAIS

Neste capítulo foi apresentado o estudo de caso para avaliação da AVS em uma simulação de um ambiente de produção de testes da Motorola. A AVS foi avaliada em comparação com a ferramenta padrão utilizada pela empresa (Jira).

O objetivo do estudo de caso foi avaliar os ganhos proporcionados pelo uso das técnicas de tratamento e processamento de dados utilizados na AVS em comparação com o sistema de busca do Jira. Com isso, foi possível comprovar a importância da utilização desses recursos para permitir que sejam realizadas melhores pesquisas entre os relatórios de erros de uma base. Através disso, visualizamos a possibilidade da diminuição da ocorrência de relatórios duplicados criados devido as deficiências proporcionadas pelos sistemas de buscas de um GRE.

6

CONCLUSÃO

Esta dissertação teve como objetivo investigar técnicas de pré-processamento e análise de relatórios de erros, com o intuito de criar um sistema de buscas descomplicado e ágil capaz de dar suporte aos GRE na mitigação das dificuldades encontradas em seus sistemas de buscas que levam a criação de relatórios de erros duplicados. Inicialmente, a partir de uma base de dados de relatórios de erros foi possível investigar, ajustar e padronizar o conteúdo dos relatórios (*i.e.*, determinando o que era ou não relevante para o domínio). Após isso, foram criados tesauros que agiram de forma imprescindível para estudo e entendimento do domínio e, além disso, essenciais para realização de buscas através de consultas textuais com maior cobertura e precisão dentro de uma base composta por documentos criados com o uso de linguagem natural.

As técnicas implementadas foram aplicadas e utilizadas em um domínio real de uma empresa privada, onde os relatórios de erros são um dos principais artefatos da linha de produção, e também, onde a existência de relatórios de erros duplicados devido às buscas imprecisas dos GRE era considerada algo relevante.

Este trabalho apresentou como o pré-processamento dos dados é fundamental para um sistema de buscas de relatórios de erros. Evidenciando também, a importância da coleta e do tratamento adequado das informações do domínio, e da criação de tesauros para a realização de buscas mais precisas e com maior cobertura dos dados.

Apesar de existirem muitas formas de um relatório de erro ser duplicado, a utilização desta abordagem demonstrou ser de grande importância colaborativa. A AVS tornou-se um contribuinte não apenas para a mitigação de duplicações, mas também para melhorias tanto na facilidade e agilidade, quanto na produtividade de um processo considerado essencial na linha de produção de uma empresa de testes.

6.1 TRABALHOS FUTUROS

Como propostas para trabalhos futuros, é possível:

- Adicionar o campo de comentários como conteúdo para as buscas. No campo de comentários de um relatório podem existir informações ainda mais relevantes a respeito da situação descrita pelo relatório, como, novas observações, detalhes mais aprofundados do erro, etc. Neste trabalho, esse campo não foi incluído nas pesquisas devido limitações na coleta dos campos da base principal. Contudo, dada a importância de informações que os comentários podem conter, este campo se torna essencial para novas avaliações.
- Aplicar a metodologia proposta em bases de diferentes domínios, variando a ordem dos processos para estudos em novos experimentos. Idiomas além do inglês podem necessitar de ajustes específicos. Com isso, será possível avaliar o comportamento da ferramenta perante diferentes bases com múltiplos idiomas.
- Criação de um sistema para automatização parcial de tesauros. Os tesauros utilizados, foram totalmente criados de forma manual, ou seja, necessitaram de muito esforço e tempo para sua elaboração. Na coleta dos termos, foi observado que em algumas situações, há termos com poucas variações em suas estruturas, além do uso constante de siglas que representam termos múltiplos. O processo de identificação desses termos pode ser feito de forma automática. Desta forma, seria possível diminuir o tempo de criação dos tesauros para qualquer domínio utilizado. Além disso, técnicas relacionadas ao contexto poderiam ser utilizadas para aprimorar a busca por termos ou expressões relacionadas nos documentos. Dessa forma seria possível contribuir para a criação de um tesauro taxonômico, capaz de relacionar itens com um processo mais inteligente.
- Utilização de um corretor ortográfico (*i.e.*, *spell checker*) relacionado a termos técnicos. Muitos termos técnicos são digitados incorretamente, em sua grande maioria por desatenção ou falta de conhecimento do usuário a cerca da escrita da palavra. Com a aplicação de um corretor ortográfico, o processo de buscas poderia se tornar mais eficiente, eliminando situações onde erros ortográficos prejudicam os resultados.

- Padronização de *templates* para relatórios de erros. Devido o uso de linguagem natural na construção dos relatórios e a diversidade de times, observamos que usuários de diferentes localidades utilizam *templates* diferentes para a elaboração de cada documento, o que, por sua vez, dificulta o processo de buscas. Em diversas ocasiões, informações importantes são omitidas ou descritas de forma incompleta devido à falta de um *template* com os campos essenciais (*i.e.*, obrigatórios ou não) para preenchimento de informações. Além disso, a falta de ordem na criação de cada relatório, faz com que no momento de uma verificação manual (*i.e.*, leitura do relatório), o usuário interprete relatórios de formas diferentes, podendo deixar escapar informações importantes.

REFERÊNCIAS

ABREU, V. S. **O Léxico na Internet: Análise de Neologismos em Comunidades do Orkut**. . In: 3º SIMPÓSIO HIPERTEXTO E TECNOLOGIAS NA EDUCAÇÃO REDES SOCIAIS E APRENDIZAGEM. Universidade Federal de Pernambuco - Núcleo de Estudos de Hipertexto e Tecnologias na Educação - Recife: 2010. Disponível em: <http://nehte.com.br/simposio/anais/Anais-Hipertexto-2010/Verena-Santos-Abreu.pdf>. Acesso em: 22 jun. 2018

AGGARWAL, C. C.; ZHAI, C. A survey of text clustering algorithms. In: AGGARWAL, C. C.; ZHAI, C. (Eds.). . **Mining Text Data**. Boston, MA: Springer US, 2012. p. 77–128.

AGGARWAL, K. et al. **Detecting duplicate bug reports with software engineering domain knowledge**. 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). **Anais**...IEEE, mar. 2015. Disponível em: <http://ieeexplore.ieee.org/document/7081831/>. Acesso em: 13 abr. 2018

ALIPOUR, A.; HINDLE, A.; STROULIA, E. **A contextual approach towards more accurate duplicate bug report detection**. 2013 10th Working Conference on Mining Software Repositories (MSR). **Anais**...IEEE, maio 2013. Disponível em: <http://ieeexplore.ieee.org/document/6624026/>. Acesso em: 26 abr. 2018

ANVIK, J.; HIEW, L.; MURPHY, G. C. **Coping with an Open Bug Repository**. Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange. **Anais**...: eclipse '05.New York, NY, USA: ACM, 2005. Disponível em: <http://doi.acm.org/10.1145/1117696.1117704>. Acesso em: 22 ago. 2018

ANVIK, J.; HIEW, L.; MURPHY, G. C. **Who Should Fix This Bug?** Proceedings of the 28th International Conference on Software Engineering. **Anais**...: ICSE '06.New York, NY, USA: ACM, 2006. Disponível em: <http://doi.acm.org/10.1145/1134285.1134336>. Acesso em: 14 set. 2018

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Recuperação de Informação: Conceitos e Tecnologia das Máquinas de Busca**. 2. ed. [s.l.] Bookman Editora, 2013.

BANERJEE, S.; CUKIC, B.; ADJEROH, D. **Automated Duplicate Bug Report Classification Using Subsequence Matching**. Proceedings of IEEE International Symposium on High Assurance Systems Engineering. **Anais**...1 out. 2012

BARROS, M.; NETTO, F.; BAIÃO, F. **Mineração da Base de Dados de Defeitos de Software**. Rio de Janeiro: DIA/UNIRIO, 2009.

BATISTA, G. E. DE A. P. A. **Pré-processamento de dados em aprendizado de máquina supervisionado**. text—[s.l.] Universidade de São Paulo, 16 maio 2003.

BONFIM, M. E. Recuperação de Documentos-Texto Usando Modelos Probabilísticos Estendidos. **Iniciação Científica Cesumar**, v. 11, n. 2, 2009.

BORG, M. et al. **A replicated study on duplicate detection: Using apache lucene to search among Android defects**. Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14. **Anais**...New York, New York, USA: ACM Press, set. 2014. Disponível em: <http://dl.acm.org/citation.cfm?doid=2652524.2652556>. Acesso em: 29 abr. 2018

BORGES, G. S. B. **INDEXAÇÃO AUTOMÁTICA DE DOCUMENTOS TEXTUAIS: PROPOSTA DE CRITÉRIOS ESSENCIAIS**. Encontro Nacional de Pesquisa em Ciência da Informação: Universidade Federal de Minas Gerais, 2009.

BRITO, E. M. Mineração de Textos: Detecção automática de sentimentos em comentários nas mídias sociais. **Projetos e Dissertações em Sistemas de Informação e Gestão do Conhecimento**, mar. 2017.

CARDOSO, Olinda Nogueira Paes. **Recuperação de Informação**. INFOCOMP, [S.l.], v. 2, n. 1, p. 33-38, nov. 2004. ISSN 1982-3363. Disponível em: <http://www.dcc.ufla.br/infocomp/index.php/INFOCOMP/article/view/46>. Acesso em: 17 mai. 2018.

CATAE, F. S. **Classificação automática de texto por meio de similaridade de palavras: um algoritmo mais eficiente.** text—[s.l.] Universidade de São Paulo, 8 jan. 2013.

CAVALCANTI, Y. C. A Bug Report Analysis and Search Tool: Improving search and analysis of duplicate bug reports. jan. 2009.

CAVALCANTI, Y. C. et al. The bug report duplication problem: an exploratory study. **Software Quality Journal**, v. 21, n. 1, p. 39–66, 2013.

CORREIA, M. F. DE B. **Recuperação de Informação**. Universidade Virtual Africana: 20 fev. 2018. Disponível em: <https://oer.avu.org/handle/123456789/653>. Acesso em: 17 maio. 2018

COSTA, J. A. F. **Classificação Automática e Análise de Dados por Redes Neurais Auto-Organizáveis**. THESIS.DOCTORAL—[s.l.] Universidade Estadual de Campinas - SP, dez. 1999.

CREMONESI, P.; KOREN, Y.; TURRIN, R. **Performance of recommender algorithms on top-N recommendation tasks**. . In: RECSYS'10 - PROCEEDINGS OF THE 4TH ACM CONFERENCE ON RECOMMENDER SYSTEMS. 1 jan. 2010

CRISTIANO, C. Gestão de defeitos. **Engenharia de Software Magazine**, n. 1, p. 76, 2007.

CUBRANIC, D.; MURPHY, G. C. **Automatic bug triage using text categorization**. . In: SEKE. Canada: 2004. Disponível em: <http://www.cs.ubc.ca/labs/spl/papers/2004/seke04-bugzilla.pdf>. Acesso em: 26 abr. 2018

DANG, Y. et al. **ReBucket: A Method for Clustering Duplicate Crash Reports Based on Call Stack Similarity**. Proceedings of the 34th International Conference on Software Engineering. **Anais**...: ICSE '12.Piscataway, NJ, USA: IEEE Press, 2012. Disponível em: <http://dl.acm.org/citation.cfm?id=2337223.2337364>. Acesso em: 3 maio. 2018

DE ÁVILA, R. L. F.; SOARES, J. M. **Uso de técnicas de pré-processamento textual e algoritmos de comparação como suporte à correção de questões dissertativas: experimentos, análises e contribuições**. : Simpósio brasileiro de informática na educação.Sociedade Brasileira de Computação, nov. 2013. Disponível em: <http://www.br-ie.org/pub/index.php/sbie/article/view/2551>. Acesso em: 18 ago. 2018

GOPALAN, R. P.; KRISHNA, A. **Duplicate bug report detection using clustering**. 2014 23rd Australian Software Engineering Conference. **Anais**...IEEE, abr. 2014. Disponível em: <http://ieeexplore.ieee.org/document/6824114/>. Acesso em: 13 abr. 2018

HINDLE, A. Stopping duplicate bug reports before they start with Continuous Querying for bug reports. ago. 2016.

INDURKHYA; DAMERAU, N.; FRED, J. **Handbook of Natural Language Processing**. [s.l.] Chapman & Hall/CRC, 2010.

JALBERT, N.; WEIMER, W. **Automated duplicate detection for bug tracking systems**. 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN). **Anais**...IEEE, jun. 2008. Disponível em: <http://ieeexplore.ieee.org/document/4630070/>. Acesso em: 13 abr. 2018

KEVIC, K. et al. **Collaborative bug triaging using textual similarities and change set analysis**. 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). **Anais**...IEEE, maio 2013. Disponível em: <http://ieeexplore.ieee.org/document/6614727/>. Acesso em: 13 abr. 2018

L. S. GASPAR, L. W., F. C. Bernardini. **Análise do uso de recuperação da informação em documentos de atendimento - Estudo de caso em bases de soluções de informática**. 2016. Acesso em: 13 abr. 2018

LAGUTINA, N. et al. **An approach to automated thesaurus construction using clusterization-based dictionary analysis**. 2015 17th Conference of Open Innovations Association (FRUCT). **Anais**...IEEE, abr. 2015. Disponível em: <http://ieeexplore.ieee.org/document/7117979/>. Acesso em: 13 abr. 2018

LAMKANFI, A. et al. **Predicting the severity of a reported bug**. 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). **Anais**...IEEE, maio 2010. Disponível em: <http://ieeexplore.ieee.org/document/5463284/>. Acesso em: 13 abr. 2018

LESKOVEC, A.; RAJARAMAN, A.; ULLMAN, J. D. **Mining of Massive Datasets**. Cambridge: Cambridge University Press, 2014.

LI, Y. H. Classification of text documents. **The Computer Journal**, v. 41, n. 8, p. 537–546, ago. 1998.

LIMSETTHO, N. et al. **Automatic Unsupervised Bug Report Categorization**. 2014 6th International Workshop on Empirical Software Engineering in Practice. **Anais**... In: 2014 6TH INTERNATIONAL WORKSHOP ON EMPIRICAL SOFTWARE ENGINEERING IN PRACTICE. nov. 2014

LIMSETTHO, N. et al. Unsupervised Bug Report Categorization Using Clustering and Labeling Algorithm. **International Journal of Software Engineering and Knowledge Engineering**, v. 26, n. 07, p. 1027–1053, 1 set. 2016.

LineageOS JIRA. Disponível em: <https://jira.lineageos.org/browse/REGRESSION-2102?jql=>. Acesso em: 30 ago. 2018.

MADUREIRA, F. A. P. **Classificação de Documentos**. THESIS.MASTER—Lisboa: Universidade Nova de Lisboa, 2009.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. **Introduction to Information Retrieval**. New York, NY, USA: Cambridge University Press, 2008.

MARKO, T.; ALEKSANDAR, S. A Survey of Bug Tracking Tools: Presentation, Analysis and Trends. 2011.

MCCALLUM, A. K. Multi-label text classification with a mixture model trained by EM. **AAAI 99**, 1999.

MINH, P. N. An Approach to Detecting Duplicate Bug Reports using N-gram Features and Cluster Chrinkage Technique. v. 4, n. 5, p. 8, 2014.

MORAL, C. et al. A survey of stemming algorithms in information retrieval. **Information Research**, mar. 2014.

MOSCATO; VON ZUBEN. **Uma Visão Geral de Clusterização de Dados**. pdf. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia368_02/topico5_02.pdf>. Acesso em: 3 maio. 2018.

MURPHY, K. P. **Machine learning: a probabilistic perspective**. Disponível em: <https://ai.google/research/pubs/pub38136>. Acesso em: 20 ago. 2018.

NAGWANI, N. K.; VERMA, D. S. Software Bug Classification using Suffix Tree Clustering (STC) Algorithm. **IJCST**, v. 2, n. 1, p. 6, 2011.

NGUYEN, A. T. et al. **A Topic-based Approach for Narrowing the Search Space of Buggy Files from a Bug Report**. Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. **Anais**...: ASE '11.Washington, DC, USA: IEEE Computer Society, 2011. Disponível em: <http://dx.doi.org/10.1109/ASE.2011.6100062>. Acesso em: 14 ago. 2018

NGUYEN, A. T. et al. **Duplicate bug report detection with a combination of information retrieval and topic modeling**. Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012. **Anais**...New York, New York, USA: ACM Press, set. 2012. Disponível em: <http://dl.acm.org/citation.cfm?doid=2351676.2351687>. Acesso em: 13 abr. 2018

OCHI, L. S.; DIAS, C. R.; SOARES, S. S. F. Clusterização em Mineração de Dados. p. 46, 2004.

PANDA, B.; SAHOO, S.; PATNAIK, S. K. A Comparative Study of Hard and Soft Clustering Using Swarm Optimization. v. 4, n. 10, p. 6, 2013.

PRIFTI, T.; BANERJEE, S.; CUKIC, B. **Detecting Bug Duplicate Reports Through Local References**. Proceedings of the 7th International Conference on Predictive Models in Software Engineering. **Anais**...: Promise '11.New York, NY, USA: ACM, 2011. Disponível em: <http://doi.acm.org/10.1145/2020390.2020398>. Acesso em: 29 ago. 2018

RAKHA, M. S.; SHANG, W.; HASSAN, A. E. Studying the needed effort for identifying duplicate issues. **Empirical Software Engineering**, v. 21, n. 5, p. 1960–1989, 1 out. 2016.

ROBERTSON, S. E.; JONES, K. S. Relevance weighting of search terms. **Journal of the American Society for Information Science**, v. 27, n. 3, p. 129–146, 1 maio 1976.

ROBERTSON, S. E.; WALKER, S.; BEAULIEU, M. Experimentation as a way of life: Okapi at TREC. **Information Processing & Management**, v. 36, n. 1, p. 95–108, 1 jan. 2000.

ROCHA, H. et al. NextBug: a Bugzilla extension for recommending similar bugs. **J Softw Eng Res Dev**, v. 3, n. 1, p. 3, dez. 2015.

ROCHA, H. et al. **An empirical study on recommendations of similar bugs**. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). **Anais**...IEEE, mar. 2016. Disponível em: <http://ieeexplore.ieee.org/document/7476629/>. Acesso em: 13 abr. 2018

RODEGHERO, P. et al. An empirical study on how expert knowledge affects bug reports. **J. Softw. Evol. and Proc.**, v. 28, n. 7, p. 542–564, jul. 2016.

RUNESON, P.; ALEXANDERSSON, M.; NYHOLM, O. **Detection of duplicate defect reports using natural language processing**. 29th International Conference on Software Engineering (ICSE'07). **Anais**...IEEE, maio 2007. Disponível em: <http://ieeexplore.ieee.org/document/4222611/>. Acesso em: 13 abr. 2018

SAHA, R. K. et al. **Improving bug localization using structured information retrieval**. 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). **Anais**...IEEE, nov. 2013. Disponível em: <http://ieeexplore.ieee.org/document/6693093/>. Acesso em: 22 abr. 2018

SCHIESSL, J. M. Descoberta de conhecimento em texto aplicada a um sistema de atendimento ao consumidor. 4 mar. 2009.

SEN, S.; GANPATI, A.; SHARMA, A. K. AN EXPLORATORY STUDY OF DUPLICATE BUG REPORTS IN OSS PROJECTS. p. 8, 2014.

SHI, Z.; KEUNG, J.; SONG, Q. **An Empirical Study of BM25 and BM25F Based Feature Location Techniques**. Proceedings of the International Workshop on Innovative Software Development Methodologies and Practices. **Anais**...: InnoSWDev 2014.New York, NY, USA: ACM, 2014. Disponível em: <http://doi.acm.org/10.1145/2666581.2666594>. Acesso em: 22 maio. 2018

SILVA, E. M. DA; SOUZA, R. R. **Comparing three different techniques to retrieve documents using multiwords expressions**. TECSI, 12 jun. 2013. Disponível em: <http://www.tecsi.fea.usp.br/contecsi/index.php/contecsi/10contecsi/paper/view/67>. Acesso em: 4 maio. 2018

SILVA JÚNIOR, V. E. DA. Uma nova abordagem do real adaboost resistente a overfitting para classificação de dados binários. ago. 2016.

SINGHAL, A. Modern Information Retrieval: A Brief Overview. **Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**, v. 24, n. 4, p. 35–42, 2001.

SOARES, F. D. A. **MINERAÇÃO DE TEXTOS NA COLETA INTELIGENTE DE DADOS NA WEB**. THESIS.MASTER—[s.l.] PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO - PUC-RIO, 2009.

SOHRAWARDI, S. J.; AZAM, I.; HOSAIN, S. **A comparative study of text classification algorithms on user submitted bug reports**. Ninth International Conference on Digital Information Management (ICDIM 2014). **Anais**...IEEE, set. 2014. Disponível em: <http://ieeexplore.ieee.org/document/6991434/>. Acesso em: 13 abr. 2018

SOUCY, P.; MINEAU, G. W. **Beyond TFIDF Weighting for Text Categorization in the Vector Space Model**. Proceedings of the 19th International Joint Conference on Artificial Intelligence. **Anais**...: IJCAI'05.San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. Disponível em: <http://dl.acm.org/citation.cfm?id=1642293.1642474>. Acesso em: 2 maio. 2018

STEIN, R.; DE BARCELOS SILVA, A. Análise assintótica de algoritmo para geração de matriz termo-documento contendo TF-IDF. jan. 2016.

SUN, C. et al. **Querying Sequential Software Engineering Data**. Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. **Anais**...: FSE 2014.New York, NY, USA: ACM, 2014. Disponível em: <http://doi.acm.org/10.1145/2635868.2635902>. Acesso em: 4 out. 2018

SUREKA, A.; JALOTE, P. **Detecting Duplicate Bug Report Using Character N-Gram-Based Features**. 2010 Asia Pacific Software Engineering Conference. **Anais**...IEEE, nov. 2010. Disponível em: <http://ieeexplore.ieee.org/document/5693213/>. Acesso em: 22 abr. 2018

SWAPNA, D.; THAMMI REDDY, K. A study of information retrieval approaches in duplicate bug detection. **Indian J Sci Technol**, v. 9, n. 43, nov. 2016.

TAMRAWI, A. et al. **Fuzzy Set and Cache-based Approach for Bug Triaging**. Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. **Anais**...: ESEC/FSE '11.New York, NY, USA: ACM, 2011. Disponível em: <http://doi.acm.org/10.1145/2025113.2025163>. Acesso em: 14 set. 2018

TIAN, Y. et al. Automated prediction of bug report priority using multi-factor analysis. **Empir Software Eng**, v. 20, n. 5, p. 1354–1383, out. 2015.

TIAN, Y.; LO, D.; LAWALL, J. **Automated construction of a software-specific word similarity database**. 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). **Anais**...IEEE, fev. 2014. Disponível em: <http://ieeexplore.ieee.org/document/6747213/>. Acesso em: 13 abr. 2018

TIAN, Y.; LO, D.; SUN, C. **Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction**. 2012 19th Working Conference on Reverse Engineering. **Anais**... In: 2012 19TH WORKING CONFERENCE ON REVERSE ENGINEERING. out. 2012

VIRGINIA, G.; NGUYEN, H. S. Investigating the effectiveness of thesaurus generated using tolerance rough set model. In: KRYSZKIEWICZ, M. et al. (Eds.). . **Foundations of intelligent systems**. Lecture notes in computer science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. v. 6804p. 705–714.

WANG, S.; LO, D. **Version history, similar report, and structure: putting them together for improved bug localization**. Proceedings of the 22nd International Conference on Program Comprehension - ICPC 2014. **Anais**...New York, New York, USA: ACM Press, jun. 2014. Disponível em: <http://dl.acm.org/citation.cfm?doid=2597008.2597148>. Acesso em: 13 abr. 2018

WANG, X. et al. **An approach to detecting duplicate bug reports using natural language and execution information**. Proceedings of the 13th international conference on Software engineering - ICSE '08. **Anais**...New York, New York, USA: ACM Press, maio 2008. Disponível em: <http://portal.acm.org/citation.cfm?doid=1368088.1368151>. Acesso em: 12 abr. 2018

WEN, M.; WU, R.; CHEUNG, S.-C. Locus: Locating bugs from software changes. 2016.

XIA, X. et al. **An empirical study of bug report field reassignment**. 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). **Anais**... In: 2014 SOFTWARE EVOLUTION WEEK - IEEE CONFERENCE ON SOFTWARE MAINTENANCE, REENGINEERING, AND REVERSE ENGINEERING (CSMR-WCRE). fev. 2014

XU, W.; LIU, X.; GONG, Y. **Document Clustering Based on Non-negative Matrix Factorization**. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval. **Anais**...: SIGIR '03.New York, NY, USA: ACM, 2003. Disponível em: <http://doi.acm.org/10.1145/860435.860485>. Acesso em: 2 maio. 2018

XUAN, J. et al. Automatic Bug Triage using Semi-Supervised Text Classification. **arXiv**, abr. 2017.

ZAMAN, A. N. K.; MATSAKIS, P.; BROWN, C. **Evaluation of stop word lists in text retrieval using Latent Semantic Indexing**. 2011 Sixth International Conference on Digital Information Management. **Anais**...IEEE, set. 2011. Disponível em: <http://ieeexplore.ieee.org/document/6093315/>. Acesso em: 13 abr. 2018

ZHANG, J. et al. A survey on bug-report analysis. **Science China Information Sciences**, v. 58, n. 2, p. 1–24, 1 fev. 2015.

ZHOU, Y. et al. **Combining text mining and data mining for bug report classification**. 2014 IEEE International Conference on Software Maintenance and Evolution. **Anais**...IEEE, set. 2014. Disponível em: <http://ieeexplore.ieee.org/document/6976097/>. Acesso em: 27 abr. 2018

ZIMMERMANN, T. et al. **Improving bug tracking systems**. 2009 31st International Conference on Software Engineering - Companion Volume. **Anais**... In: 2009 31ST INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - COMPANION VOLUME. maio 2009

ZIMMERMANN, T. et al. What Makes a Good Bug Report? **IEEE Transactions on Software Engineering**, v. 36, n. 5, p. 618–643, set. 2010.

ZOU, J. et al. Automated Duplicate Bug Report Detection Using Multi-Factor Analysis. **IEICE Trans Inf Syst**, v. E99.D, n. 7, p. 1762–1775, 2016.