Construção de um classificador automático de severidade de *bugs* para sistemas *open source*

Cláudio Ribeiro de Sousa

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2016

Cláudio Ribeiro de Sousa

Construção de um classificador automático de severidade de *bugs* para sistemas *open source*

Dissertação de Mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Engenharia de Software

Orientador: Prof. Dr. Marcelo de Almeida Maia

Uberlândia
2016

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

S725c
2016

Sousa, Cláudio Ribeiro de, 1988-
Construção de um classificador automático de severidade de bugs para sistemas open source / Cláudio Ribeiro de Sousa. - 2016.
74 f. : il.

Orientador: .
Coorientador: Marcelo de Almeida Maia.
Dissertação (mestrado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.
Inclui bibliografia.

1. Computação - Teses. 2. Software - Qualidade - Controle - Teses. 3. Software - Testes - Teses. I. Maia, Marcelo de Almeida. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU:

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Os abaixo assinados, por meio deste, certificam que leram e recomendam para a Faculdade de Computação a aceitação da dissertação intitulada "**Construção de um classificador automático de severidade de *bugs* para sistemas *open source***" por **Cláudio Ribeiro de Sousa** como parte dos requisitos exigidos para a obtenção do título de **Mestre em Ciência da Computação.**

Uberlândia, 11 de março de 2016

Orientador: _____
Prof. Dr. Marcelo de Almeida Maia
Universidade Federal de Uberlândia

Banca Examinadora:

_____
Prof. Dr. Stéphane Julia
Universidade Federal de Uberlândia

_____
Prof. Dr. Mark Alan Junho Song
Pontifícia Universidade Católica de Minas Gerais

À todos aqueles que me apoiaram e me deram forças, MUITO OBRIGADO!

Agradecimentos

Em primeiro lugar, à Deus, por sempre iluminar meu caminho.

Aos meus pais, Sebastião e Osvanilda, por todo apoio, atenção, compreensão e educação dado durante toda a minha vida. Todas as minhas conquistas só fazem sentido por que são para vocês.

Ao meu irmão, Adriano, pelos momentos de companheirismo e pela amizade verdadeira de toda a vida.

A minha amada companheira Camila, pelo amor, atenção, companheirismo e compreensão durante todos os nossos dias juntos. Você é meu porto seguro e minha inspiração diária.

Aos amigos, pela convivência, pelos momentos de alegria e relaxamento e pela cooperação para superar todos obstáculos e avançar as etapas deste trabalho.

Por fim, ao meu orientador, Prof. Marcelo de Almeida Maia, pela oportunidade e desafios proporcionados. Sua orientação me permitiu evoluir constantemente e superar os desafios. Ficarei sempre grato.

Todos os nossos sonhos podem se tornar realidade se tivermos a coragem de persegui-los. (Walt Disney)

Resumo

A análise de *bugs* de *software* é uma das atividades mais importantes na Qualidade de *Software*. A rápida e correta implementação do reparo necessário tem influência tanto para os desenvolvedores, que devem deixar o *software* funcionando plenamente, quanto para os usuários, que precisam executar suas tarefas diárias. Neste contexto, caso haja incorreta caracterização no relato dos *bugs*, podem ocorrer situações indesejadas. Um dos principais fatores a serem atribuídos ao *bug* no ato de seu relato inicial é severidade, que diz respeito à urgência da correção daquele problema. Diante deste cenário, identificou-se em conjuntos de dados *bugs* extraídos de cinco sistemas *open source* (*Apache, Eclipse, Kernel, Mozilla e Open Office*), que há uma distribuição não uniforme dos *bugs* com relação às severidades existentes nesta amostra, o que é um indício inicial de má classificação. Nos dados analisados existe uma taxa de cerca de 85% de *bugs* sendo classificados com apenas a severidade normal. Logo, esta taxa de classificação pode influenciar negativamente no contexto do desenvolvimento do *software*, em que o *bug* mal classificado pode ser alocado para um desenvolvedor com pouca experiência para poder resolvê-lo e assim, a correção do mesmo pode demorar, ou ainda gerar uma implementação incorreta. Vários trabalhos na literatura tem desconsiderado os *bugs* normais, trabalhando apenas com a porção de *bugs* considerados severos ou não severos inicialmente. Este trabalho teve como principal objetivo investigar esta porção dos dados, com finalidade de identificar se a severidade normal reflete a real urgência de correção do *bug*, investigar se existem *bugs* (classificados inicialmente como normais) que poderiam ser classificados com outra severidade, além de avaliar se há impactos para os desenvolvedores neste sentido. Para isso, foi desenvolvido um classificador automático, que baseou-se em três algoritmos (*Näive Bayes, Max Ent* e *Winnow*) para aferir se a severidade normal está correta para os *bugs* assim categorizados inicialmente. Os algoritmos apresentaram acurácia de cerca de 80%, e mostraram que entre 21% e 36% dos *bugs* deveriam ter sido classificados de outra forma (dependendo do algoritmo), o que representa algo entre 70.000 e 130.000 *bugs* da amostra coletada.

Palavras-chave: Qualidade de *software*, *bug trackers*, repositório de *bugs*, severi-

dade de *bugs*

Abstract

Software bug analysis is one of the most important activities in Software Quality. The rapid and correct implementation of the necessary repair influence both developers, who must leave the fully functioning software, and users, who need to perform their daily tasks. In this context, if there is an incorrect classification of bugs, there may be unwanted situations. One of the main factors to be assigned bugs in the act of its initial report is severity, which lives up to the urgency of correcting that problem. In this scenario, we identified in datasets with data extracted from five open source systems (Apache, Eclipse, Kernel, Mozilla and Open Office), that there is an irregular distribution of bugs with respect to existing severities, which is an early sign of misclassification. In the dataset analyzed, exists a rate of about 85% bugs being ranked with normal severity. Therefore, this classification rate can have a negative influence on software development context, where the misclassified bug can be allocated to a developer with little experience to solve it and thus the correction of the same may take longer, or even generate a incorrect implementation. Several studies in the literature have disregarded the normal bugs, working only with the portion of bugs considered severe or not severe initially. This work aimed to investigate this portion of the data, with the purpose of identifying whether the normal severity reflects the real impact and urgency, to investigate if there are bugs (initially classified as normal) that could be classified with other severity, and to assess if there are impacts for developers in this context. For this, an automatic classifier was developed, which was based on three algorithms (Näive Bayes, Max Ent and Winnow) to assess if normal severity is correct for the bugs categorized initially with this severity. The algorithms presented accuracy of about 80%, and showed that between 21% and 36% of the bugs should have been classified differently (depending on the algorithm), which represents somewhere between 70,000 and 130,000 bugs of the dataset.

Keywords: Software quality, bug trackers, bug repository, bug severity

Lista de ilustrações

Lista de tabelas

Sumário

Capítulo 1

Introdução

Nos dias atuais, o desenvolvimento de sistemas geralmente esbarra com problemas relativos à qualidade e tempo. O mercado de *software*, e tecnologia da informação em geral, exige, cada vez mais, sistemas rápidos, eficientes e com alto nível de qualidade, face à alta dependência do homem destes sistemas.

Um problema comum em *softwares* é que eles apresentam falhas (MYERS; SANDLER, 2004). Um *bug* (como é comumente chamada pela comunidade de desenvolvimento de *software*) é identificado quando um serviço prestado pelo *software* se desvia do serviço correto que ele deveria prestar (LYU, 1996). Segundo um estudo realizado pelo *National Institute of Standards and Technology*, *bugs* de *software* custam cerca de 60 bilhões de dólares por ano na indústria norte-americana (SINGH; KAUR; MALHOTRA, 2010). Isso pode ser explicado, pelo fato de que a prevenção de falhas, antes mesmo da introdução do defeito que as origina, no código-fonte, é uma tarefa bastante difícil, se não impossível, considerando que os programadores estão suscetíveis a cometerem erros durante a execução de suas tarefas (AVIZIENIS et al., 2004). Há também estudos indicando que o custo da correção de uma falha cresce 10 vezes a cada etapa que se passa do ciclo de vida do desenvolvimento de um *software* (RIOS et al., 2012), o que evidencia ainda mais a importância de tratar o assunto com cautela. Desta forma, começamos a perceber a importância de estarmos atentos aos procedimentos relativos à qualidade dos sistemas produzidos, visto que um dos problemas mais sérios gerados pela não preocupação com este seguimento, é o financeiro.

Existem duas principais atividades que ocorrem durante o ciclo de vida de um *software* que focam principalmente nos *bugs,* sendo a primeira delas o teste, que consiste no processo de executar um *software* com o objetivo de encontrá-los. Dentro da atividade de teste, já deparamos com uma das etapas iniciais do processo que é de suma importância para este trabalho, o relato dos *bugs*. É nesta etapa que os desenvolvedores ou testadores irão descrever o que aconteceu de errado na aplicação e então solicitar sua respectiva correção. Porém, como veremos, nem sempre este relato acontece de forma apropriada,

especialmente quanto à classificação da urgência de correção daquele *bug*, usualmente relatada no campo severidade.

A segunda atividade é a manutenção, que é conhecida como uma atividade durante a qual ocorrem modificações em um ou mais artefatos construídos no desenvolvimento de *software*, buscando mantê-lo disponível, corrigir seus *bugs*, melhorar seu desempenho e/ou adequá-lo aos requisitos novos ou modificá-los, de acordo com as necessidades de seu usuário (PIGOSKI, 1996). Algumas estatísticas revelaram que no início dos anos noventa, muitas organizações alocaram, no mínimo, 50% de seus recursos financeiros em manutenção de *software* (BETTENBURG et al., 2007), enquanto na década passada, a manutenção de software tem chegado a 70% dos custos de um sistema (CHATURVEDI; SINGH, 2012). Podemos perceber que esta etapa pode ser impactada diretamente pela atividade de teste. Como o relato do problema é um dos primeiros passos realizados para que seja feita a devida correção, quando não há um relato feito de forma adequada, a implementação do reparo necessário pode ser influenciada negativamente. O principal obstáculo acontece quando um *bug* não é relatado com a urgência correta; assim, caso ele necessite de atenção mais rápida, sua correção pode ser feita por um desenvolvedor menos experiente ou com menos habilidade técnica, visto que foi relatado de forma indevida. Desta forma, o reparo do *bug* relatado incorretamente pode atrasar ou gerar uma implementação imperfeita, gerando transtornos maiores, inclusive financeiros. Este trabalho foca especialmente em ajudar a melhorar o entendimento geral da comunidade sobre os *bugs* classificados como normais e apresentar como sua classificação pode influenciar no contexto de desenvolvimento profissional.

Para efetuar uma melhor gerência dos *bugs* encontrados, existem sistemas *on-line*, denominados *bug-trackers*, geralmente integrados aos repositórios de *software*, que servem como plataforma central para relato e monitoramento dos *bugs* pertinentes ao respectivo desenvolvimento (SERRANO; CIORDIA, 2005). Nestes sistemas, além de poder inserir novos *bugs*, os usuários podem obter informações importantes para determinar a qualidade do produto que está sendo desenvolvido, como por exemplo: componentes mais defeituosos do sistema, quantidade de *bugs* severos, entre outros.

Neste trabalho, todo o conjunto de dados foi coletado da ferramenta de gestão de *bugs open source Bugzilla*[1]. A maior vantagem do *Bugzilla* é que ele funciona como um repositório completo, permitindo que as pessoas descrevam, classifiquem, comentem e selecionem *bugs* para efetuar as devidas correções. Assim, procurou-se encontrar um conjunto de dados *bugs* relatados para diferentes situações e aplicações. Ao final desta busca, foram selecionados dados de cinco projetos *open source*: *Apache*[2]*, Eclipse*[3]*, Kernel*[4]*,*

[1] http://www.bugzilla.org/
[2] http://bugs.apache.org/
[3] http://bugs.eclipse.org/bugs/
[4] http://bugzilla.kernel.org

Mozilla[5] e *Open Office*[6]. Para o desenvolvimento deste trabalho, foram coletados 354.532 *bugs* reportados entre 2010 e 2014, já corrigidos.

Um dos fatores existentes nos *bugs* descritos neste repositório, e que chama a atenção no contexto deste trabalho, é a severidade atribuída a cada *bug* relatado. A severidade pode ser definida como o impacto que determinado problema causa no funcionamento de um sistema. Quanto maior o impacto e o dano causado, maior será então sua severidade e, consequentemente, maior será a urgência de sua correção. O *Bugzilla* disponibiliza aos seus usuários sete níveis distintos de severidades, disponíveis em (Eclipse Foundation, 2015b), que são eles:

- *Blocker*: falha que causa bloqueio das atividades de desenvolvimento e/ou teste do sistema.
- *Critical*: falha que causa quedas, perda de dados, ou problemas com uso excessivo de memória.
- *Major*: falha que causa perda de funções principais do sistema.
- *Normal*: falha comum no sistema, que não tem impacto tão grave, apresentam apenas alguma perda de funcionalidade sob circunstâncias específicas.
- *Minor*: falha que causa perda de funcionalidades com menor relevância ou problemas fáceis de serem contornados.
- *Trivial*: falhas que causam problemas simples de serem resolvidos, como uma grafia ou cor errada.
- *Enhancement*: não são falhas no sistema, mas sim modificações solicitadas no sistema com intuito de aperfeiçoá-lo.

Através de observações feitas do conjunto de dados, identificou-se que cerca de 85% dos *bugs* relatados foram classificados com a severidade Normal. Além disso, alguns estudos descartam totalmente estes *bugs* e trabalham apenas com aqueles que denominam como severos (composição feita das severidades *Blocker*, *Critical* e *Major)* e não severos (compostos pelos *bugs Trivial* e *Minor*) (LAMKANFI et al., 2010). Desta forma, este trabalho teve como principal foco estudar estes *bugs* classificados como normais, com intuito de identificar se as suas severidades refletem a real urgência de correção e, caso isso não seja verdade, qual a real classificação. Para isso, os conceitos de grupos *bugs* severos e não severos, já mencionados, foram utilizados, porém acrescidos do grupo dos normais. Desta forma, o intuito do trabalho é desenvolver um classificador automático, que irá analisar característcas de cada *bug* e definir a qual grupo ele realmente pertence.

[5] http://bugzilla.mozilla.org
[6] http://bz.apache.org/ooo/

1.1 Problema de pesquisa

A classificação correta das severidades atribuídas a cada um dos *bugs* pode ser considerada como fator fundamental no tempo que a falha resultante levará para ser corrigida. Uma classificação incorreta deste campo, que representa o nível de exigência que se espera do seu reparo, pode levar o mesmo a ser alocado como tarefa de correção de um desenvolvedor menos experiente, que por sua vez, tenderá a demorar mais para terminar a tarefa, necessitará de ajuda de outros desenvolvedores para finalizá-la, ou ainda, implementará uma correção inadequada ou incompleta. Isto acontece porque os desenvolvedores estão em constante pressão para manter seu *software* disponível e em pleno funcionamento. Para isso, a alocação de desenvolvedores mais experientes na correção de *bugs* mais complexos e impactantes se faz necessária, visto que, caso o *bug* seja atrelado a um desenvolvedor que não tenha conhecimento adequado, o processo de correção poderá ter problemas.

Alguns estudos selecionam conjuntos de dados de *bugs* para serem estudados e utilizam-se da severidade como critério de seleção daqueles que serão ou não relevantes para suas pesquisas (TIAN; LO; SUN, 2013; LAMKANFI et al., 2010; SHARMA et al., 2012; LAMKANFI et al., 2011; TIAN; LO; SUN, 2012; GARCIA; SHIHAB, 2014). Algo comum nos estudos citados é o fato de que eles excluem *bugs* inicialmente relatados com a severidade normal. Segundo eles, os *bugs* normais representam uma grande área cinzenta no conjunto de dados, contendo informações confusas. Além disso mencionam que esta é a classificação padrão e, por isso, quando o testador não sabe como classificar, ele deixa o *bug* como normal, bastando estes argumentos para removê-los dos seus estudos. Ao avaliar estes trabalhos é importante ressaltar duas situações: os estudos mostram como é difícil e árdua a tarefa de classificar um novo *bug*, especialmente quando se trata da severidade normal, e há uma lacuna criada devido a essa exclusão dos *bugs* classificados como normais dos trabalhos.

Porém, ao iniciarmos um estudo neste grupo de *bugs* normais, foi identificado que esta classificação não está sendo feita de forma adequada. Em nosso conjunto de dados, cerca de 85% dos *bugs* estão classificados com a severidade normal, e existem indícios de que esta não é a real severidade de todos os *bugs* assim classificados. Podemos ver que alguns deles estão claramente classificados de forma errônea. Tomemos como exemplo o *bug* número 16140, extraído do conjunto de dados do sistema do *Kernel*. Este *bug* demorou 1283,49 dias para ter sua correção enfim implementada, possui 17 anexos e 50 comentários, feitos por 15 pessoas distintas. O relator do *bug*, voltou a comentar cinco vezes após sua primeira mensagem. Em sua descrição resumida, existem três palavras que são comumente encontradas em *bugs* classificados como severos (no mesmo sistema *Kernel*) e ele afeta um Produto e um Componente do sistema que estão na lista daqueles mais problemáticos (com maior número de *bugs*). Adicionando a todas estas informações, temos que este *bug*

bloqueia a correção de um outro, de número 15310, ou seja, a correção do *bug* bloqueado só pode iniciar após a correção do primeiro.

Baseado em todas as informações apresentadas, temos fortes indícios de que o *bug* 16140 não foi classificado corretamente pela pessoa que o relatou, já que o mesmo tem características que pertecem ao grupos dos *bugs* severos. Além disso, verifica-se que houve a necessidade de outras pessoas intervirem para que a correção do mesmo fosse enfim implementada e o outro problema no sistema então pudesse ter sua correção iniciada. Diante do que foi exposto, nota-se que, caso este sistema fosse estudado por alguém que levasse em consideração apenas *bugs s*everos e não severos, ele estaria trabalhando com uma amostra menos representativa. Isto foi objeto de estudo de (Ripon K. Saha et al., 2015), porém a base de dados utilizada nesta pesquisa consistiu de apenas alguns dos componentes do *Eclipse,* diferentemente da usada neste trabalho, que analisou cinco diferentes ferramentas *open source*. Por fim, também é natural percebermos que a correção não foi alocada para o desenvolvedor mais apropriado, visto que o mesmo necessitou de auxílio de outras pessoas para enfim efetivar a correção do mesmo. Além disso, o tema da escolha de qual desenvolvedor irá ser alocado para cada tarefa, especialmente de correção de *bugs*, é estudado e demonstrado que não é uma atividade simples (BORTIS; HOEK, 2013; XUAN et al., 2012; ANVIK; HIEW; MURPHY, 2006).

Neste trabalho foram estudados estes *bugs* considerados "normais", visando identificar indícios de má classificação dos mesmos e seus impactos. Desta forma, foram levantadas três hipóteses:

H1: As classificações das severidades dos *bugs* está distribuída de forma não uniforme;

H2: Existem *bugs* classificados com a severidade normal, a qual não reflete a real severidade dos mesmos;

H3: A má classificação das severidades dos *bugs* geram maiores discussões entre desenvolvedores durante suas correções.

Para averiguar tais hipóteses, foram estudadas as distribuições e proporções dos *bugs* relatados em um repositório de cinco diferentes sistemas, buscando entender melhor o cenário atual. Este estudo objetivou a criação de uma metodologia para o desenvolvimento de um classificador automático das severidades dos *bugs,* que, por sua vez, é baseado em três diferentes algoritmos de classificação (*Näive Bayes, Max Ent* e *Winnow*). Os resultados deste classificador têm por objetivo obter uma melhor assertividade nos relatos de *bugs* e auxiliar o desenvolvedor em suas tarefas prioritárias.

1.2 Objetivos e contribuição

Como mencionado anteriormente, o desenvolvedor é cobrado e pressionado atualmente devido à alta dependência da sociedade em relação aos sistemas computacionais. Além disso, vários sistemas possuem similares, desta forma, se o desenvolvedor demorar para realizar as devidas correções em seus produtos, ele pode perder usuários para seus concorrentes. Diante disso, ele não pode postergar a correção de problemas graves existentes nos *softwares*. Além do que, caso esta correção se prolongue, o custo poderá ser maior e mais difícil de se realizar.

Diante deste cenário, este trabalho procurou auxiliar o desenvolvedor, ao propor uma solução que melhora a classificação de *bugs* e, desta forma, permitir a este desenvolvedor focar na correção de questões mais urgentes.

O principal objetivo deste trabalho é criar uma metodologia para desenvolvimento de um classificador automático de *bugs*, visando identificar características que permeiam os mesmos, para indicar a real severidade deles.

Para avaliar tal objetivo, foram estabelecidas as seguintes perguntas de pesquisa:

1. Como estão distribuídos os *bugs* com relação às severidades? Esta pergunta nos proporcionará identificar a disparidade nas classificações e avaliar a primeira hipótese.

2. Os *bugs* incorretamente classificados podem ser reclassificados automaticamente? Com base nas características dos *bugs* classificados nos grupos de severos e não severos, poderemos extrair informações que nos auxiliem a realizar tal reclassificação. Desta forma, podemos avaliar a segunda hipótese.

3. Os *bugs* incorretamente classificados trouxeram algum impacto para o trabalho do desenvolvedor? Aqui tentamos identificar se realmente houve impacto prático da incorreta classificação, avaliando o que foi proposto na terceira hipótese.

1.3 Estrutura da dissertação

Além deste presente capítulo introdutório, este trabalho apresenta-se desenvolvido e documentado dentro da seguinte estrutura organizacional:

Capítulo 2: Referencial Teórico.

Nesse capítulo serão apresentados os conceitos sobre a ferramenta *Mallet*, que foi utilizada para desenvolvimento do classificador automático deste trabalho, bem como para extração de características importantes dos *bugs* analisados. Também serão explicados resumidamente os algoritmos de classificação utilizados para aferir a severidade automaticamente.

Capítulo 3: Metodologia.

Nesse capítulo será apresentada a metodologia, constituída em 10 etapas, usada para conduzir a pesquisa com o conjunto de dados coletado. As 10 etapas foram divididas em três fases, visando responder as perguntas de pesquisa levantadas e avaliar as hipóteses. A primeira fase, denominada Caracterização dos *Bugs*, compreende as três primeiras etapas deste trabalho. Nela, foi feito um refinamento dos *bugs*, ou seja, seleção apenas daqueles já finalizados no repositório, além de uma análise das características principais dos *bugs* e, por fim, um levantamento de como estão distribuídos os *bugs* quanto às severidades. A segunda fase, denominada Classificação Automática, compreende as etapas 4, 5, 6, 7 e 8 deste trabalho. Nela foram observadas as principais características dos *bugs,* para que pudesse ser feita, inicialmente, uma classificação manual, depois uma validação externa e por fim a classificação automatizada. Na última fase, denominada Pós-validação e análises finais, os resultados obtidos pela fase anterior foram novamente validados, bem como as conclusões finais quanto aos impactos no contexto do desenvolvedor puderam ser avaliados. Esta última fase compreendeu as etapas 9 e 10 deste trabalho

Capítulo 4: Resultados e Discussão.

Nesse capítulo serão apresentados e discutidos os resultados da aplicação da pesquisa, dentre as três fases estabelecidas na metodologia. Para isto, serão expostos tabelas, figuras e informações que apresentam as quantidades de *bugs* e suas distribuições perante severidades, quantidade de *bugs* reclassificados e validados manualmente, bem como os resultados da classificação automática com os três algoritmos utilizados: *Näive Bayes, Max Ent* e *Winnow*.

Capítulo 5: Trabalhos Relacionados.

Aqui serão apresentados trabalhos relacionados com esta pesquisa e que motivaram o desenvolvimento da mesma, bem como será feita uma comparação dos resultados aqui obtidos com outras pesquisas.

Capítulo 6: Considerações Finais.

Este capítulo apresentará as conclusões sobre a pesquisa, destacando seus principais resultados e os trabalhos futuros que podem ser gerados. Além disso, serão expostas as ameaças à validade da pesquisa, bem como detalhadas as formas utilizadas para sanar seus impactos.

Capítulo 2

Referencial Teórico

A construção do classificador automático deste trabalho utilizou a ferramenta *Mallet* para seu desenvolvimento; diante disso, na primeira seção deste capítulo, o funcionamento da ferramenta será detalhado. Na segunda seção, serão apresentados os conceitos dos algoritmos de classificação utilizados.

2.1 Ferramenta *Mallet*

O *Mallet* é uma ferramenta, baseada em *Java*, que pode ser utilizada para processamento estatístico de linguagem natural, classificação de documentos, *clustering*, modelagem de tópicos, extração de informações, dentre outras aplicações para efetuar análises textuais. Para realizar tais análises textuais, o *Mallet* utiliza conceitos de aprendizado de máquina, transformando os documentos de texto em representações numéricas, visando um processamento mais eficiente.

No próprio *website* da ferramenta (MCCALLUM, 2015), é possível efetuar o *download* da versão mais atual do *Mallet*. Para instalá-lo, basta extrair o arquivo baixado e posteriormente alterar as variáveis de ambiente do sistema para incluir a da aplicação. Logo que a ferramenta está adequadamente instalada no computador, ela pode ser utilizada por meio de linha de comando.

Para efetuar a classificação de um grupo de dados, duas etapas simples devem ser executadas. A primeira delas consiste na importação dos dados de treinamento, que devem estar listados em um arquivo de texto. A segunda etapa do processo é a de classificação, que por sua vez deve ser executada em três procedimentos.

O primeiro procedimento da etapa de classificação é o treinamento. Para realizar o treinamento, deve ser selecionado o arquivo de texto importado para esta tarefa na primeira etapa. Após selecionado, através da linha de comando, o usuário executa um comando que trata aquele arquivo como o treinamento para o classificador automático.

Em seguida, é necessário escolher qual algoritmo de classificação será utilizado. Por padrão, o algoritmo *Näive Bayes* é o escolhido, porém, caso o usuário queira trocar, ele pode selecionar outro algoritmo desejado, também por meio da linha de comando. Após realizados os dois procedimentos, então pode ser feita a classificação automática. Para isso, o usuário deve utilizar de três arquivos, sendo dois deles gerados pela própria ferramenta: o primeiro consiste no resultado do treinamento realizado e o segundo pelo algoritmo selecionado. O último arquivo é aquele que contém todos os itens a serem classificados automaticamente, que devem ser colocados todos em um arquivo de texto para serem processados, também via linha de comando. Caso o usuário deseje a classificação por meio de outro algoritmo, o processo deve ser repetido. Ao final, os resultados podem ser extraídos em planilhas no formato *Comma Separated Values* (CSV), para análise de seus resultados.

Neste trabalho, os arquivos a serem classificados eram compostos dos dados de cada um dos *bugs* a serem classificados, listados um por cada linha do arquivo de texto. Eles puderam ser identificados pelos seus códigos identificadores, e continham as informações relativas aos seus produtos, componentes e sumários descritivos.

Além da classificação automática dos *bugs*, a ferramenta *Mallet* também foi utilizada para extração de termos mais frequentes nos *bugs* classificados como severos e naqueles classificados como não severos inicialmente. Para isso, em cada um dos sistemas estudados, foram criados dois arquivos de texto. O primeiro continha todos os sumários descritivos dos *bugs* relatados como severos, enquanto o segundo dos não severos. A seguir, são apresentados trechos de sumários descritivos utilizados para esta análise em um dos sistemas estudados, o *Kernel*:

A. Exemplo de trecho do arquivo com sumário dos *bugs* severos:

Summary

Problem mounting btrfs-volume during boot

[ivb] WARNING at drivers/gpu/drm/i915/intel_pm.c:5997 intel_display_power_put

B. Exemplo de trecho do arquivo com sumário dos *bugs* não severos:

Summary

Can't register new acpi battery -> No battery life on laptop

poll() fd negation trick doesn't work for fd 0

Com estes dois arquivos gerados manualmente, para cada um dos sistemas, foi utilizada a opção de modelagem de tópicos oferecida pelo *Mallet*. Para realizar tal extração de termos, o *Mallet* associa não só os termos mais comuns, mas também aqueles que possuem certa semelhança. Por exemplo, as palavras *"pet"* e *"dog"* são mapeadas para um

tópico baseando-se na semelhança de ocorrência, ambas contextualizadas como animal de estimação.

Assim como na classificação automática, o primeiro passo para analisar estes arquivos foi realizar a importação dos mesmos para dentro da ferramenta via linha de comando. Com os dados importados, basta utilizar os comandos para extração de tópicos nos textos. Após toda a análise textual, o *Mallet* devolve um arquivo de resposta. Este arquivo contém a listagem dos termos que a ferramenta julgou como sendo os mais relevantes e frequentes dentre os existentes no arquivo de texto importado (GRAHAM; WEINGART; MILLIGAN, 2012).

2.2 Algoritmos de Classificação

Para realizar as classificações automáticas deste trabalho, foram selecionados três algoritmos disponibilizados pela ferramenta *Mallet*. Cada algoritmo possui uma estratégia de reconhecimento do padrão de classificação estabelecido pelos atributos do classificador. Estas estratégias estão relacionadas a estudos estatísticos contidos no assunto Aprendizado de Máquina. Para esta pesquisa, foram utilizados os seguintes algoritmos: *Max Entropy, Naïve Bayes* e *Winnow*. A ferramenta dispõe de outros algoritmos para classificação automática, no entanto, estes foram os selecionados, pois foram os únicos que trouxeram aderência aos dados analisados neste estudo. Os demais algoritmos, também foram testados, como por exemplo Árvore de Decisão, porém, não conseguiram realizar as devidas classificações dos *bugs*, apresentando, ao final de suas análises, resultados que não permitiam classificar os dados em nenhum dos três níveis de severidade utilizados neste trabalho, por isso foram descartados. A seguir é apresentado, resumidamente, o funcionamento dos algoritmos selecionados para este estudo.

O modelo estatístico da Entropia Máxima, aqui chamado *Max Ent*, executa o cálculo das somas de probabilidades de cada *feature* ser igual a um, levando em consideração a construção de modelos estocásticos através de uma função não linear (MCCALLUM; FREITAG; PEREIRA, 2000). Diante disso, foi verificado para cada *bug* qual o valor aferido pelo classificador para cada um dos três níveis de severidades (severo, não severo e normal). O nível que obteve maior valor foi então considerado como a nova classificação. Em (ZHANG, 2015) é apresentado um conjunto de ferramentas para estudo aprofundado sobre este algoritmo.

O algoritmo *Naïve Bayes*, que é baseado na hipótese Bayesiana (MCCALLUM; NIGAM, 1998), define que dado um documento de teste, a classificação deste é executada através do cálculo da probabilidade de cada classe considerando o maior valor desta probabilidade dentre as amostras de entrada. O processo é aplicado iterativamente para todas as amostras do documento de teste. No caso deste trabalho, o algoritmo atribuiu uma

probabilidade para cada nível de severidade que os *bugs* poderiam ser classificados (severo, não severo e normal). Da mesma forma que o algoritmo anterior, o nível de severidade que obteve maior valor de probabilidade foi o considerado como novo nível de classificação do *bug* em análise.

O algoritmo *Winnow* é um modelo linear de classificação baseado em uma função booleana de predição (GOLDING; ROTH, 1999). No processo de predição do algoritmo *Winnow*, a função preditiva de classificação atribui valores 0 e 1 às amostras. Além disso, esta função está acompanhada por um conjunto de pesos para cada atributo que, ao final, soma os valores (com seus pesos) para cada um dos níveis. Assim como nos casos dos algoritmos anteriores, o maior valor aferido dentre os níveis de severidade foi o fator determinante para indicar qual a severidade dos *bugs* analisados.

Capítulo 3

Metodologia

Este trabalho tem como objetivo principal a construção do classificador automático para severidades de *bugs.* O intuito da criação deste classificador é auxiliar na identificação de quais *bugs* relatados como normais deveriam ter sidos classificados de outra maneira, o que pode ajudar o trabalho dos desenvolvedores e pesquisadores. Desta forma, serão expostas a seguir as etapas deste trabalho e os experimentos realizados.

3.1 Etapas de Pesquisa

O estudo realizado nesta pesquisa consistiu em um total de 10 etapas. Sendo que as cinco primeiras etapas tiveram como foco principal a seleção e o entendimento dos dados utilizados e as cinco últimas etapas foram destinadas às manipulações e análises do conjunto de dados. A seguir, são apresentadas cada uma das etapas citadas:

1. Refinamento do conjunto de dados: etapa em que foi feita busca por *bugs* finalizados e com sua correção já implementada no *bug-tracker* de cada ferramenta aqui estudada. Foram acessados todos os *websites* de cada aplicação (Apache Software Foundation, 2015; Eclipse Foundation, 2015a; Apache Software Open Office Foundation, 2015; Mozilla Foundation, 2015; Linux Kernel Organization, 2015) onde foram coletados os *bugs* em formato de tabela CSV.

2. Entendimento das características dos *bugs*: buscou-se entender como é feito o relato de um *bug*, bem como quais campos que os relatores devem preencher. Desta forma, procurou-se compreender manualmente quais seriam as características relevantes de cada *bug* e que seriam úteis para a classificação automática.

3. Análise de distribuição dos *bugs*: procurou-se identificar indícios de má classificação dos *bugs* coletados, baseado nas características coletadas na etapa anterior. A principal característica verificada foi a severidade do *bug*, que em resumo, define a complexidade e o grau de impacto que o mesmo tem na aplicação. Neste contexto,

observou-se como estão distribuídos os *bugs* dentre os níveis definidos pelo repositório e pelos grupos de severos, não severos e normais.

4. Identificação de Produtos e Componentes mais defeituosos: foram identificados os Produtos e Componentes com maior incidência de *bugs* em cada ferramenta analisada. Estas foram consideradas as áreas que merecem maior atenção dos sistemas, já que são críticas por serem as mais defeituosas e, consequentemente, mais utilizadas pelos usuários;

5. Identificação de termos mais recorrentes: usando a ferramenta *Mallet*, foi feito um levantamento daquelas palavras mais comuns nos *bugs* considerados severos e nos *bugs* considerados não severos, para que estes sejam utilizados na reclassificação como fator crucial para identificar se o *bug* pertence ao grupo dos severos ou não severos;

6. Reclassificação manual de *bugs*: foi estabelecida uma metodologia, onde após serem selecionados aleatoriamente alguns *bugs*, os mesmos foram submetidos e avaliados com relação à sua severidade. Desta forma, conseguiu-se produzir um conjunto de treinamento para o classificador automático, com situações onde *bugs* classificados incialmente como normais, foram reclassificados dentro dos grupos de severos e não severos;

7. Validação externa: foram selecionadas alunos de um curso da área de tecnologia da região do triângulo mineiro (pessoas externas a este trabalho) para que elas pudessem, após preparação, validar a reclassificação manual feita anteriormente apenas pelo autor;

8. Reclassificação automática: utilizando a ferramenta *Mallet*, e dos dados reclassificados manualmente nas etapas anteriores, foi feita uma reclassificação automática de todos os *bugs* relatados inicialmente como normais. Foram utilizados três algoritmos de classificação distintos, visando obter maior segurança nos resultados obtidos;

9. Pós validação manual: foram selecionados alguns *bugs* de cada ferramenta (reclassificados automaticamente) para que pudesse ser feita uma validação posterior à reclassificação automática. Esta etapa procurou aferir qual a taxa de acerto dos algoritmos escolhidos;

10. Coleta e análise dos resultados: etapa final onde todas as informações foram observadas e averiguadas para identificar se realmente houve impacto da má classificação no processo de correção dos *bugs*.

As etapas 1, 2 e 3 foram agrupadas na primeira fase deste estudo denominada Caracterização dos *Bugs*. Esta fase teve como intuito principal avaliar a distribuição de

bugs dentre os grupos de severidades criados, respondendo a primeira pergunta proposta e permitindo analisar a hipótese 1 deste trabalho para identificar se há realmente uma distribuição não uniforme.

Já as etapas 4, 5, 6, 7 e 8 foram agrupadas para a fase denominada Classificação Automática. Ela visou identificar se a severidade normal reflete o real impacto dos *bugs* e o desenvolvimento do classificador automático, permitindo avaliar a hipótese 2 e responder a segunda pergunta de pesquisa.

*A*s demais etapas (9 e 10) integraram a fase Pós-validação e análises finais, e tiveram como foco analisar o que fora proposto na terceira hipótese deste trabalho, ou seja, qual é o impacto da má classificação na correção e análise de *bugs.* Além disso, também foram avaliadas as respectivas contribuições desta pesquisa, especialmente para melhor alocação dos desenvolvedores que corrigem os *bugs.*

3.2 Caracterização dos *Bugs*

Nesta subseção serão detalhadas as etapas do trabalho que compreendem a primeira fase deste estudo, que teve como foco analisar as características dos *bugs* e identificar as suas respectivas distribuições para avaliar primeira hipótese proposta e responder a primeira pergunta de pesquisa.

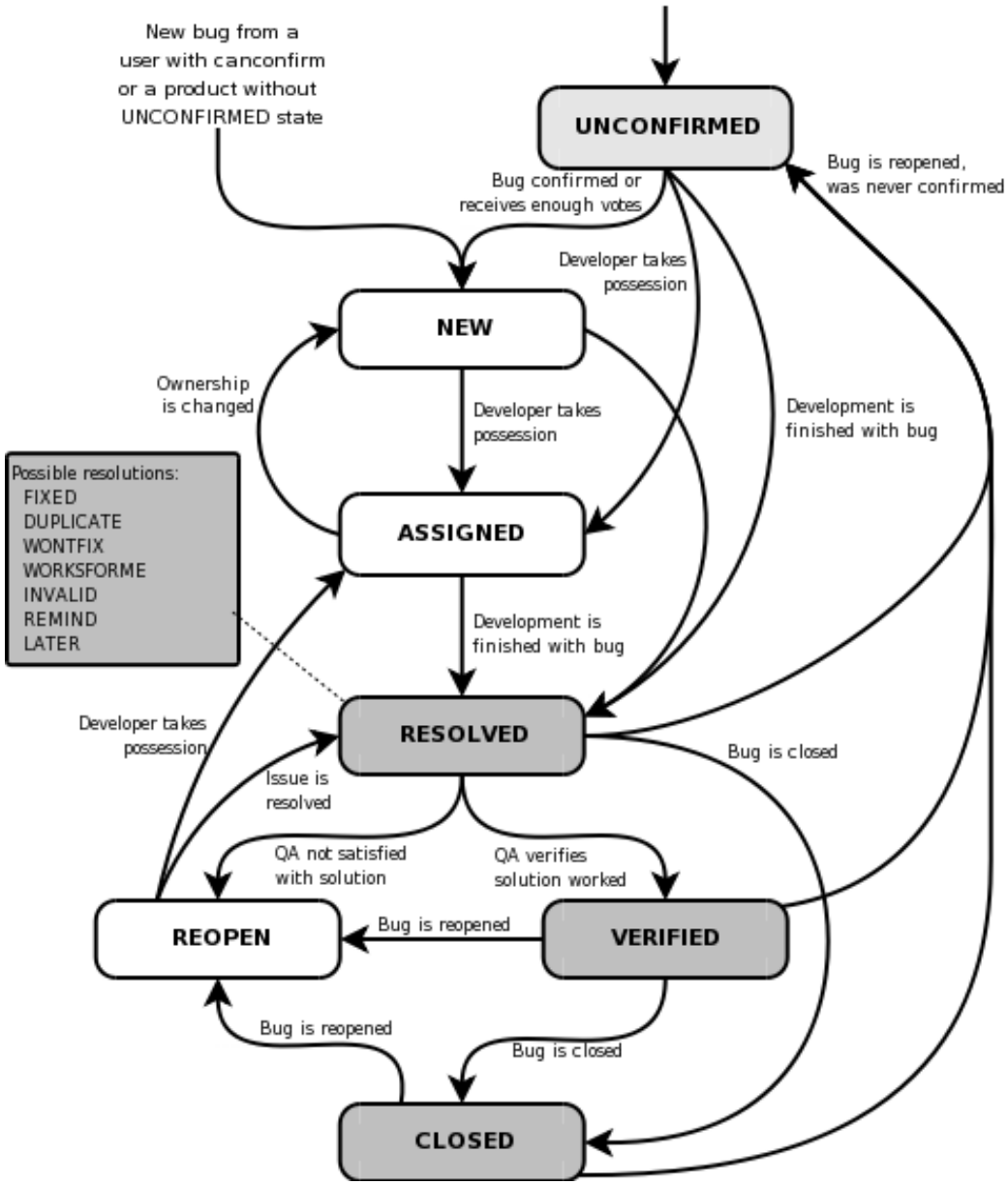
3.2.1 Refinamento do conjunto de dados

Como primeira etapa da pesquisa, foi realizada uma análise nos *bugs* coletados do conjunto de ferramentas *Apache, Eclipse, Kernel, Mozilla e Open Office*. Esta análise teve como objetivo principal refinar os dados, ou seja, extrair do *Bugzilla* apenas *bugs* úteis para serem analisados (no contexto deste trabalho seriam úteis apenas *bugs* já finalizados e com sua correção também já implementada no respectivo sistema) e buscar entender como é feita a classificação e a descrição dos *bugs* reportados. Ao final desta etapa foi possível ter uma caracterização dos *bugs* a serem trabalhados, com suas respectivas quantidades e proporções.

O refinamento do conjunto de dados coletado ocorreu por meio da busca e seleção dos *bugs* já resolvidos, verificados ou fechados nas páginas oficiais dos repositórios de cada sistema na *web*. Como todos os sistemas utilizam o *Bugzilla*, foram filtrados na ferramenta todos os *bugs* que possuíam no campo *Status* os valores *Resolved, Verified e Closed*, e campo *Resolution* igual à *Fixed*, conforme pode ser visto na Figura 1. Desta forma, conseguimos selecionar apenas aqueles *bugs* já tratados e corrigidos pelas equipes de desenvolvimento de cada ferramenta. Também é importante destacar que foram selecionados apenas *bugs* solucionados entre janeiro de 2010 e dezembro de 2014. Este filtro foi realizado para que se pudesse utilizar nesta pesquisa *bugs* mais atuais dos sistemas escolhidos. Ao analisar

outros trabalhos (Philip J. Guo et al., 2010; Ripon K. Saha et al., 2015), identificou-se que eram utilizados conjuntos de dados, relativos a *bugs*, mais antigos. Assim sendo, o atual trabalho apresenta um estudo com dados novos, podendo apresentar mudanças e/ou cenários diferentes dos já estudados, no entanto, a comparação de fato com os demais trabalhos não foi feita durante o estudo conduzido por este trabalho.

Figura 1 – Ciclo de vida de um *bug,* (BARNSON, 2006)



3.2.2 Entendimento das características dos *bugs*

Em seguida, na segunda etapa, buscou-se analisar como é o processo para reportar um novo *bug*, bem como quais campos são necessários preencher e ter conhecimento para solicitar uma correção no *software*. Como resultado desta análise, observou-se que na ferramenta de gerenciamento de *bugs Bugzilla*, para reportar um novo *bug*, a pessoa

inicialmente deve estar cadastrada no sistema e possuir um usuário de acesso e uma senha. Como todos os sistemas analisados neste trabalho são livres, qualquer pessoa pode solicitar o cadastro e reportar uma inconsistência encontrada durante o uso do sistema em questão. Após ingressar na aplicação, e antes de relatar a inconsistência encontrada, é apresentado ao usuário algumas informações para instruí-lo quanto ao procedimento de relato de *bugs*, em seguida, o usuário então deve preencher os seguintes campos do novo *bug*: Produto, Componente, Versão, Severidade, Prioridade, Sumário, Plataforma, Sistema Operacional e Descrição. Ao analisar a literatura correlata (TIAN; LO; SUN, 2013; LAMKANFI et al., 2010; SHARMA et al., 2012; LAMKANFI et al., 2011; TIAN; LO; SUN, 2012; GARCIA; SHIHAB, 2014)), identificou-se que alguns estudos focaram na análise de campos que em princípio influenciam diretamente na urgência da correção dos *bugs*, que são: severidade e prioridade. O primeiro campo está relacionado com o impacto que o problema gera na aplicação como um todo, variando de uma simples correção de cor ou texto à um bloqueio completo das funcionalidades do sistema, e está presente em todos os *bugs* coletados. Já o segundo campo define qual *bug* vai ser corrigido primeiramente e qual pode ter sua correção postergada. Os dois campos certamente se relacionam, no entanto, no conjunto de dados observado, não existe prioridade para todos os *bugs*, e por isso, o campo prioridade não foi objeto de estudo. Porém, como atualmente a concorrência no mercado de *software* é grande, visto que várias aplicações executam tarefas semelhantes ou até iguais, a correção dos *bugs* mais impactantes no sistema acabam sendo correções mais prioritárias, pois estas bloqueiam, total ou parcialmente, o uso das ferramentas. Caso o bloqueio persista por muito tempo, aquele produto pode perder espaço no mercado para uma ferramenta similar e emergente. Assim, definimos que a severidade seria o ponto chave a ser analisado e os campos Produto, Componente e Sumário descritivo poderiam ser utilizados como auxiliares na classificação, visto que, todos os bugs possuem estes campos preenchidos e, pode-se extrair informações relevantes dos mesmos, como: quais produtos e componentes são os mais defeituosos e quais termos são mais comuns e relevantes nos sumários descritivos.

3.2.3 Análise da distribuição dos *bugs*

Nesta terceira etapa foi feita uma análise quanto à proporção de *bugs* coletados para cada severidade existente no *Bugzilla*. O primeiro fato que chamou a atenção nesta etapa foi a distribuição desproporcional de *bugs* dentre as severidades, corroborando com a hipótese 1. No *Mozilla*, 88,3% da amostra é construída de *bugs* com severidade normal. Já no *Eclipse*, esta proporção consiste em 81,6,4% da amostra, enquanto no *Kernel* são 80,4% e 71,4% no *Apache*. A única ferramenta que diverge do cenário exposto foi o *Open Office*, onde apenas 34,5% dos *bugs* são classificados como normais (nesta amostra, 58,4% dos *bugs* foi classificada como não severo). Considerando toda a amostra (de todos os sistemas analisados), temos que 85,7% dos *bugs* foram classificados com a severidade Normal, o que representa uma desproporção significativa nas classificações. Além disso, podemos ver que,

caso excluíssemos os *bugs* normais de nosso estudo, como feito naqueles trabalhos citados no parágrafo anterior, nosso estudo contemplaria menos de 15% do total de situações relatadas nas ferramentas, o que poderia resultar em análises, observações e informações incompletas e que não representam a realidade como um todo.

3.3 Classificação Automática

Nesta subseção serão detalhadas as etapas do trabalho que compreendem a segunda fase deste estudo, que teve como foco analisar e extrair as informações importantes para efetuar a classificação dos *bugs*, inicialmente manual (para produção de dados para treinamento), e posteriormente, a classificação automatizada, visando avaliar segunda hipótese proposta e responder a segunda pergunta de pesquisa.

3.3.1 Identificação de Produtos e Componentes mais defeituosos

Diante do fato de que a maioria de *bugs* reportados em quase todos os sistemas analisados fora classificado como normal, procurou-se avaliar todos estes *bugs* visando identificar se os mesmos não possuíam características que os levariam a uma nova classificação. A literatura é vaga quanto à definição de *bugs* normais, pois os definem como problemas regulares ou perda de algumas funcionalidades. No entanto, em relação às demais definições, há uma maior clareza. Por exemplo, *bugs* severos geralmente são definidos como sendo situações onde as atividades do desenvolvimento e/ou teste ficam totalmente bloqueadas, quedas, perdas de memória e funções majoritárias dos sistemas, ou seja, no caso do *Bugzilla*, são considerados severos os *bugs* classificados como *Blocker, Critical ou Major*. Por outro lado, *bugs* não severos são definidos como sendo mal funcionamento de partes não principais do sistema, problemas cosméticos, ou seja, aqueles problemas mais simples, geralmente relacionados com cores distintas do solicitado, texto mal formatado, escrito incorretamente ou com alinhamento incorretos, por exemplo, ou ainda outras situações de baixa complexidade. Neste caso, podemos associar como não severos os *bugs* classificados no *Bugzilla* como *Trivial* e *Minor*. É importante ressaltar que estas são as definições oficiais, encontradas em (Eclipse Foundation, 2015b).

Diante destas definições, buscou-se analisar os *bugs* normais visando encontrar características neles que se aproximassem de uma severidade maior ou uma severidade menor, ou seja, identificar nestes dados informações que os caracterizem de forma diferente da realizada inicialmente. Para realizar esta reclassificação, partiu-se inicialmente do Princípio de Pareto, já relatado em outros trabalhos (KUO; HUANG, 2010; IQBAL; RIZWAN, 2009; HUANG; KUO; LUAN, 2014). Nestes estudos, o princípio aponta que 20% de todos módulos, componentes e partes dos sistemas estudados representam 80% das funcionalidades principais dos mesmos. Desta forma, nosso estudo avaliou se o Princípio de

Pareto se aplicava em dois campos importantes presentes nos relatos dos *bugs*: Produto e Componente. Para isso, foram listados 20% dos Produtos e Componentes mais defeituosos, ou que continham maior quantidade de *bugs*. Isto é, foram calculadas quais as quantidades de *bugs* que cada Produto e Componente possui, e selecionados 20% do total daqueles com maior número de *bugs*. Verificou-se posteriormente, que estes 20% selecionados, representavam cerca de 80% ou mais da quantidade total de *bugs* relatados. Esta lista encontra-se disponível no Apêndice A deste trabalho.

3.3.2 Identificação de termos mais recorrentes

A quinta etapa do trabalho, consistiu na procura de termos relevantes encontrados nos *bugs* classificados como severos *(blocker, critical ou major)* ou não severos *(trivial ou minor)*. Para realizar esta etapa, a ferramenta *Mallet* foi utilizada. Nela pode ser feito um levantamento dos principais termos mais recorrentes nos *bugs* relatados de uma forma automática. Para realizar este levantamento de termos, todos os sumários resumidos dos *bugs* severos e todos dos *bugs* não severos foram colocados em dois arquivos de texto separados (para cada ferramenta). Após esta separação dos dados, os mesmos foram submetidos à ferramenta, que extraiu as informações desejadas. No entanto, após a ferramenta apontar os termos, foi necessário realizar uma apuração nas palavras que estavam repetidas ou contidas tanto nos conjuntos de severos e não severos, visando obter apenas aquelas exclusivas para cada grupo. Aqueles termos que estavam presentes em ambos os grupos foram descartados. Logo, foram criados os grupos de palavras úteis para auxiliar na definição dos casos de má classificação, e em qual grupo aquele *bug* deveria se encaixar. Então, chegou-se ao conjunto de termos disponível no Apêndice B deste trabalho.

3.3.3 Reclassificação manual

A construção de um classificador automático requer a criação de um conjunto de teste para que o algoritmo de aprendizagem use-o como referência. Desta forma, realizou-se nesta sexta etapa, uma classificação manual de uma amostra dos *bugs* existentes. Assim sendo, foi executado o procedimento a seguir para avaliar os *bugs* normais, visando identificar sua real severidade (as explicações estão no decorrer do texto):

- Seleção de 500 *bugs* de cada sistema que compõe o conjunto de dados;
- Identificação se os *bugs* pertenciam ao grupo dos Produtos mais afetados em cada um dos sistemas;
- Identificação se os *bugs* pertenciam ao grupo dos Componentes mais afetados em cada um dos sistemas;
- Identificação se os *bugs* continham termos mais comuns do grupo de severos;

- Identificação se os bugs continham termos mais comuns do grupo não severos;
- Avaliação manual e individual de cada *bug*, por meio da observação em cada *bug* da presença das características que puderam ser extraídas nas etapas anteriores.

A identificação de Produtos e Componentes mais afetados nos sistemas tem sua relevância já que são as principais partes dos sistemas. Assim, os *bugs* que afetam tais partes podem tender a não serem normais, pois afetam áreas mais defeituosas. Já a presença dos termos, severos ou não severos, em um *bug* classificado como normal, pode indicar que o mesmo fora mal classificado, e sugerir se o mesmo deve ser reclassificado para outro nível. Logo, a associação destes fatores, resultou na forma utilizada para reclassificar manualmente os *bugs*.

A avaliação manual visou buscar *bugs* que possuíam características mais próximas dos severos ou dos não severos. Desta forma, poderíamos identificar quais *bugs* possuem características que o distinguem do padrão. Assim, para realizar a reclassificação, foi feita a seguinte análise, obtida a partir do conhecimento empírico adquirido durante o estudo, em cima de cada um dos *bugs*:

- Produto E Componente relevante + Termo(s) severo(s): Severo
- Produto OU Componente relevante + Termos severos: Severo
- Presença de três ou mais termos severos: Severo
- Produto OU Componente relevante + Termo(s) não severo(s): Não Severo
- Apenas Termo(s) não severo(s): Não severo
- Produto OU Componente relevante apenas: Normal
- Apenas um termo severo: Normal

Os critérios acima definidos foram estabelecidos para análise visando selecionar aqueles casos onde há maior tendência de má classificação. Assim, para reclassificar um *bug* para severo, precisava-se ter uma confiança maior para alteração da classificação (visto que esta deve realmente representar uma situação de maior urgência); desta forma, somente aqueles que tiveram pelo menos três fatores indicando esta alteração foram reclassificados, como por exemplo: produto relevante e presença de dois termos severos. Já para os casos dos não severos, a presença de termos não severos era suficiente. Os casos onde o *bug* afetava produtos e/ou componentes relevantes ao mesmo tempo somente, mas não houve presença de nenhum termo para indicar outro nível de severidade, tiveram suas classificações mantidas como normal.

No entanto, além dos casos descritos acima, procurou-se analisar o contexto de relato do *bug*, por meio das mensagens descritas nos sumários de cada um. Desta forma, casos onde ocorreram perda de memória, quedas sucessivas e bloqueio de atividades, foram considerados como severos e correção de linguagem, cores ou alinhamentos, foram considerados como não severos. Esta análise ocorreu exclusivamente naqueles casos onde houve conflitos ou não se pode determinar claramente a severidade do *bug,* mas o contexto indicava esta necessidade. Por exemplo, quando um *bug* possui em seu sumário termos considerados severos e não severos ao mesmo tempo, neste caso, o contexto foi levado em conta. Ao final desta etapa, conseguimos obter uma taxa de cerca de 37% de *bugs* classificados incorretamente.

3.3.4 Validação externa

Para certificar e dar mais confiança aos resultados da aplicação da metodologia de reclassificação dos *bugs* normais desenvolvida, na etapa sete, foi feito um experimento envolvendo alunos matriculados no curso superior de graduação em Tecnologia em Análise e Desenvolvimento de Sistemas ofertado pelo Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro, da cidade de Uberaba/MG. Este experimento visou submeter alguns *bugs* reclassificados manualmente aos alunos, para que, após serem instruídos quanto ao trabalho e assunto abordado, os mesmos pudessem realizar suas próprias classificações e posteriormente, estas novas classificações, serem confrontadas com as realizadas anteriormente. Esta etapa teve como foco certificar que a metodologia poderia ser compreendida e aplicada por outras pessoas, bem como aferir a acurácia da classificação manual da etapa anterior.

Como requisito para participar do experimento foi exigido conhecimento intermediário ou avançado da língua inglesa, bem como o aluno estar cursando ou já ter concluído as disciplinas Engenharia de *Software* e Qualidade de *Software*, previstas na matriz curricular do curso.

Cinco alunos foram selecionados para participar do experimento, que consistiu de sete fases, descritas a seguir:

- Fase 1: Explicação sobre o trabalho realizado em um mestrado acadêmico, bem como suas exigências, restrições e regras.

- Fase 2: Explicação sobre o funcionamento da ferramenta *Bugzilla* e a respeito da composição do conjunto de dados utilizado neste trabalho.

- Fase 3: Explicação da metodologia de análise e reclassificação a ser realizada nos *bugs* selecionados.

- Fase 4: Alunos realizaram uma análise prévia, reclassificando três *bugs* selecionados de cada sistema.
- Fase 5: Acompanhamento do autor junto com os alunos, visando certificar o entendimento da metodologia, explicar detalhes necessários e avaliar classificações realizadas na fase anterior.
- Fase 6: Alunos realizaram análise de 10 *bugs* reclassificados pelo autor em cada um dos sistemas, totalizando 50 *bugs*.
- Fase 7: Levantamento das respostas dos alunos e comparação com os resultados aferidos pelo autor, visando identificar qual a taxa de classificações fora igual e qual a taxa de distinção.

Ao final, após a participação dos alunos, pode-se registrar que, em geral, os resultados apresentam 80% de concordância, o que é um valor considerado satisfatório neste contexto.

3.3.5 Reclassificação automática

Na etapa oito, antepenúltima a ser realizada, logo após o estudo realizado com os alunos, foram construídos três classificadores automáticos para indicação de severidade dos *bugs* restantes. Cada classificador utilizou um algoritmo de classificação diferente, sendo eles: *Näive Bayes, Winnow e Max Ent*. Sendo que foram utilizadas as implementações fornecidas pela ferramenta *Mallet* para construção das classificações desejadas. A base de dados classificada anteriormente de forma manual, para cada um dos sistemas aqui estudados, foi utilizada como base de treinamento para os classificadores. Identificou-se que existem entre 70.000 e 130.000 *bugs,* aproximadamente, classificados incorretamente (dependendo do algoritmo).

Assim que as classificações foram feitas, de forma automática, os resultados dos três algoritmos foram comparados. Com a comparação, surgiram quatro classes distintas de resultados, sendo elas:

1. *Bug* classificado da mesma forma nos três algoritmos;
2. *Bug* classificado da mesma forma nos algoritmos *Näive Bayes* e *Max Ent*;
3. *Bug* classificado da mesma forma nos algoritmos *Näive Bayes* e *Winnow*;
4. *Bug* classificado da mesma forma nos algoritmos *Winnow* e *Max Ent*.

Diante deste cenário, primeiramente, ficou evidente o quanto a classificação da severidade de um *bug* é uma tarefa árdua. Além disso, evidencia-se o quão suscetível a

erros estão as pessoas que têm pouco ou nenhum conhecimento do repositório de *bugs* e do sistema em desenvolvimento como um todo. Estas classes tiveram como objetivo estabelecer duas linhas de classificação possíveis: a primeira mais segura, onde os três algoritmos tiveram seus resultados iguais e uma segunda, ainda relevante, porém mais relaxada, onde apenas dois algoritmos tiveram resultados iguais. Para esta segunda situação, necessitou-se observar quais dos três cenários foi o melhor caso.

3.4 Pós-validação e análises finais

Nesta subseção serão detalhadas as duas últimas etapas do trabalho que compreendem a terceira fase deste estudo, que teve como foco analisar os resultados obtidos na fase anterior, para avaliá-los e identificar se a classificação automática foi feita adequadamente. Enfim, foi possível avaliar a terceira hipótese proposta e responder a terceira pergunta de pesquisa.

3.4.1 Pós-validação manual

Na pós-validação manual dos *bugs* classificados automaticamente, penúltima etapa deste trabalho, foram selecionados 600 exemplos no total, para reclassificação manual. Para cada uma das classes explicitadas na última etapa da fase anterior foram selecionados 150 *bugs*, sendo 30 para cada um dos sistemas. Deste 30 *bugs*, 10 foram reclassificados automaticamente como severos, 10 reclassificados automaticamente como não severos e 10 mantiveram a classificação como normal. Aqui, o intuito foi certificar que o classificador automático realmente seguiu o padrão e a metodologia de classificação estabelecida. Além disso, foi importante para identificar a taxa de acertos (entre bugs classificados automaticamente e pós-validados manualmente), visando definir a acurácia do procedimento realizado.

3.4.2 Coleta e análise dos resultados

Por fim, na décima e última etapa do trabalho, todas as informações obtidas foram analisadas para identificar se as contribuições da pesquisa realizada foram válidas e positivas. Assim sendo, foram observados o número de comentários registrados em uma amostra contendo *bugs* reclassificados como severos e outros que continuaram, mesmo após reclassificação, como normais. Além da análise do número de comentários, foram selecionados alguns casos de *bugs* reclassificados como severos para que se pudesse averiguar o teor das informações relatadas nestes comentários, com objetivo de verificar se o desenvolvedor alocado para a correção enfrentou problemas para efetivar o reparo do *bug*.

Capítulo 4

Resultados e Discussão

Neste capítulo serão mostrados e analisados os resultados obtidos. A seguir, os resultados serão apresentados seguindo a ordem de execução das respectivas fases e etapas. Na Seção 4.1, serão apresentados os resultados referentes à caracterização dos *bugs* (que corresponde às etapas 1, 2 e 3). A Seção 4.2, trará os resultados da classificação automática (que compreende as etapas 4, 5, 6, 7 e 8). Por fim, a Seção 4.3, trará observações da pós validação dos resultados e análises finais (etapas 9 e 10).

4.1 Caracterização dos *Bugs*

A primeira etapa deste trabalho consistiu em um refinamento dos *bugs* de cada sistema, visando selecionar apenas aqueles que já estavam finalizados e corrigidos. Foi necessário acessar cada um dos respositórios e efetuar buscas apenas pelos casos mencionados. Este refinamento foi importante, pois os repositórios contém *bugs* que ainda estão sendo corrigidos, ou que aguardam a intervenção de algum desenvolvedor. Como resultado da seleção, cada repositório forneceu uma planilha CSV contendo os *bugs* selecionados. Desta forma, Figura 2 apresenta como a busca é feita no repositório e a Tabela 1 apresenta as quantidade de *bugs* que pode ser coletada de cada um dos sistemas.

Figura 2 – Seleção de *bugs* no repositório do *Mozilla*

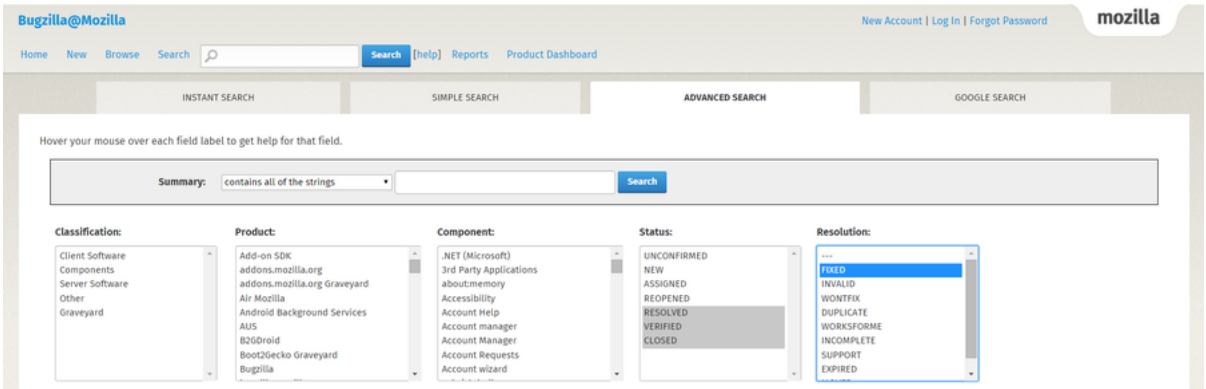


Tabela 1 – Quantidade total de Bugs do conjunto de dados

Sistema	**Bugs**
Apache	4131 (1,16%)
Eclipse	85708 (24,17%)
Kernel	2694 (0,76%)
Mozilla	256695 (72,40%)
Open Office	5304 (1,51%)
TOTAL	354532 (100%)

Em seguida, na segunda etapa, foram estudados estes *bugs* visando encontrar as principais características dos relatos feitos, que no caso deste trabalho foram: a severidade, o produto, o componente e o sumário descritivo. As planilhas geradas foram úteis, pois permitiam visualizar todos os atributos que os *bugs* possuem (código identificador, sumário descrititvo, produto, componente, sistema operacional, versão, severidade, prioridade, número de comentários, data de abertura, data de fechamento e *hardware*). Assim, observou-se que a severidade é o campo que merece maior atenção, pois reflete a urgência na correção. Isto fica evidenciado, pois em vários casos, as prioridades dos mesmos nem foram relatadas (no caso do *Mozilla*, o maior sistema analisado, 192563 *bugs* não tinham prioridade relatada). Não foi possível identificar com precisão o motivo da ausência do campo prioridade em tantos casos, porém este cenário pode indicar aquelas situações onde o usuário não soube qual prioridade associar ao *bug* e então preferiu deixar o campo em branco. Percebeu-se também, que os campos relativos aos produtos, componentes e sumários descritivos eram relevantes, pois, além de estarem relatados por completo em todos os *bugs*, eles continham descritivos textuais que poderiam ser utilizados para efetuar a classificação automática. O número de comentários também foi identificado como um dos campos importantes, e foi selecionado para ser utilizado nas etapas finais do trabalho. Para os demais campos também faltaram dados, ou seja, alguns *bugs* continham os dados e outros não, por isso não foram utilizados.

A severidade foi então definida como fator principal para análise, pois tem impacto direto na correção. Em cima desta característica, a terceira etapa procurou identificar a distribuição dos *bugs* dentre os grupos de severidade encontrados (importante ressaltar que as severidades *Blocker, Critical e Major* foram agrupadas no nível Severo, enquanto as severidades *Trivial e Minor* foram agrupadas no nível Não Severo. Além do grupo Normal). Foram então listados, manualmente, todos os *bugs* para cada um dos grupos de severidades estabelecidos. A Tabela 2 mostra a distribuição dos *bugs* por nível de severidade (Severo, Não severo e Normal).

Sumário para Pergunta de Pesquisa 1: Os resultados chamaram a atenção por apresentarem grande discrepância entre as distribuições dos *bugs*. Com 85,7% do total

de *bugs* classificados com a severidade Normal, identificamos um indício inicial da má classificação presente nos dados coletados. A hipótese 1 deste trabalho pretendia avaliar se os *bugs* não estavam distribuídos de forma uniforme entre os três níveis de severidade, logo, com esta alta taxa de *bugs* classificados como normais, vimos que a hipótese é verdadeira na prática. Além disso, a distribuição dos *bugs* dentre os outros níveis de severidades existentes também pode ser aferida, sendo 10,2% para os *bugs* severos, 4,1% para os *bugs* não severos, além dos 85,7%. Assim a primeira pergunta de pesquisa deste trabalho, que pretendia levantar esta distribuição, também pode ser respondida.

Tabela 2 – Quantidade de Bugs por nível de severidade

Sistema	**Severos**	**Não Severos**	**Normais**
Apache	514 (15,5%)	437 (13,1%)	2374 (71,4%)
Eclipse	9842 (13,2%)	3891 (5,2%)	60747 (81,6%)
Kernel	447 (16,8%)	74 (2,8%)	2145 (80,4%)
Mozilla	23110 (9,2%)	6236 (2,5%)	222476 (88,3%)
Open Office	381 (7,1%)	3132 (58,4%)	1850 (34,5%)
TOTAL	34294 (10,2%)	13770 (4,1%)	289592 (85,7%)

4.2 Classificação automática

A primeira etapa desta fase consistiu no levantamento manual dos Produtos e Componentes mais defeituosos dos sistemas. Para isso, foram utilizadas as planilhas geradas pelo *Bugzilla* na etapa um. Nestas planilhas, por meio de ordenação e listagem, foram selecionados aqueles produtos, e em seguida os componentes, que tiveram maior número de *bugs*, dentro de uma margem de 20% do total. Após este levantamento, observou-se que, os produtos listados do *Mozilla* são afetados por 83,22% de todos os *bugs*, enquanto no *Eclipse* esse valor é de 78,67%, no *Apache* 77%, no *Kernel* 72,83% e, por fim, no *Open Office*, 61,95%. Já com relação aos componentes, eles representam 91,74% dos *bugs* do *Open Office*, 88,24% dos *bugs* do *Eclipse*, 85,31% do *Apache*, 84,97% no *Mozilla* e 78,4% no *Kernel*. O apêndice A apresenta todos os Produtos e Componentes mais defeituosos e o conjunto de termos mais recorrentes dentre os níveis severo e não severo. Ambas informações foram importantes para definição da metodologia deste trabalho e posterior reclassificação, visto que já separaram os dados em grupos que são mais defeituosos, logo merecem maior atenção, e aqueles menos defeituosos, que possuem baixa complexidade.

Na quinta etapa, todos os sumários descritivos dos *bugs* severos foram colocados em um arquivo de texto, assim como os sumários dos *bugs* não severos, para cada um dos sistemas. Estes arquivos por sua vez, foram submetidos à ferramenta *Mallet*, que então pode listar os principais termos (ou palavras) existentes em cada um dos grupos. Após o resultado do *Mallet*, foi necessária uma revisão manual nestes termos, pois alguns

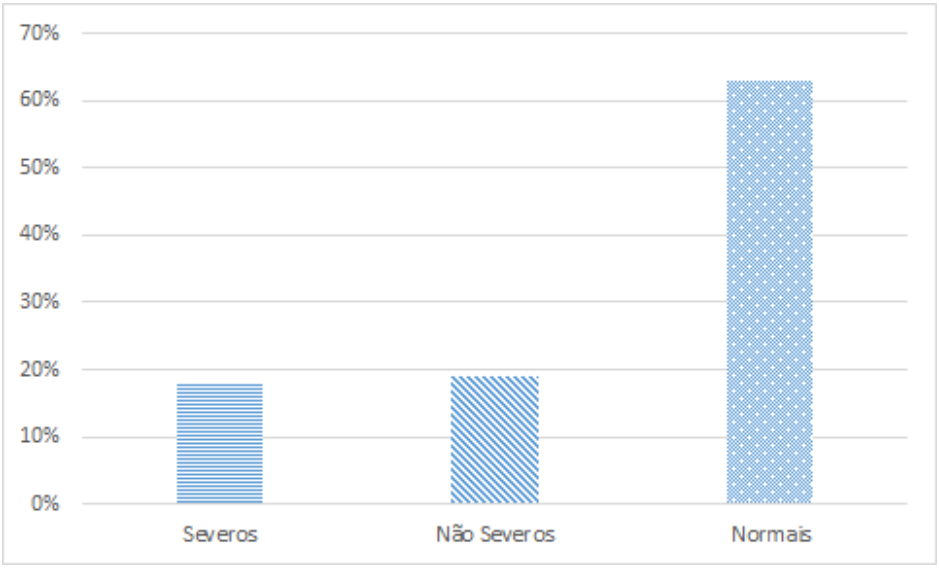
apareceram tanto na lista de severos, quanto na lista de não severos. Desta forma, estes termos em duplicidade foram removidos das listas. A relevância destes termos se dá no fato de que eles foram utilizados como uma das fontes para decidir se o *bug* mal classificado era severo ou não severo. A listagem destes termos encontra-se disponível no Apêndice B deste trabalho.

Na etapa seis foram analisados manualmente cerca de 2500 *bugs*, sendo escolhidos aleatoriamente em torno de 500 de cada um dos sistemas estudados. Esta análise consistiu na separação/reclassificação dos *bugs* entre os grupos: severo, normal e não severo. Os resultados encontrados foram os seguintes:

Tabela 3 – Resultado quantitativo da reclassificação manual dos bugs por sistema

Sistema	Severos	Não Severos	Normais
Apache	81 (16,04%)	65 (12,87%)	359 (71,09%)
Eclipse	58 (11,48%)	81 (16,04%)	366 (72,48%)
Kernel	122 (24,16%)	85 (16,83%)	298 (59,01%)
Mozilla	71 (14,06%)	112 (22,18%)	322 (63,76%)
Open Office	123 (24,36%)	130 (23,76%)	252 (51,88%)
TOTAL	455 (18,01%)	473 (18,74%)	1597 (63,25%)

Figura 3 – Distribuição de *Bugs* após reclassificação manual



É importante destacar que ao todo, 928 *bugs* (455 severos e 473 não severos) tiveram suas classificações alteradas, ou seja, foram classificados diferente de normal, o que representa cerca de 37% (18% severos e 19% não severos) dos *bugs* analisados manualmente. Logo, temos mais um indício que aponta para a má classificação dos *bugs* coletados e já apontamentos que levam a confirmar a hipótese 2 deste trabalho, ao indicar que existe uma porção dos *bugs* que não reflete a real severidade do mesmo .

Após esta etapa, os dados foram submetidos à uma validação com pessoas externas, para confirmação da metodologia. Diante disso, foram selecionados cinco alunos do curso superior em Tecnologia em Análise e Desenvolvimento de Sistemas, oferecido pelo Instituto Federal de Educação, Ciência e Tecnologia do Triângulo Mineiro (IFTM), campus Uberaba Parque Tecnológico. A seleção dos alunos consistiu na exigência de que possuíssem nível de inglês intermediário ou avançado e que já tivessem cursado as disciplinas Engenharia de *Software* e Qualidade de *Software*, previstas para os 3º e 4º períodos do referido curso.

O trabalho de validação propriamente dito consistiu em, inicialmente, realizarem uma avaliação supervisionada de *bugs* (onde foram selecionados três *bugs* por sistema). Nesta fase, procurou-se verificar como foi o entendimento dos alunos quanto às explicações do autor sobre o trabalho e metodologia desenvolvida. Assim sendo, ao final desta fase inicial, obteve-se 85,33% de concordância entre os *bugs* classificados pelo autor e os *bugs* classificados pelos alunos. Os dados completos desta fase inicial, podem ser vistos na Tabela 4.

Tabela 4 – Quantidades de acertos por aluno na fase inicial

Aluno	**Quantidade de Acertos**
Aluno 1	15 (100%)
Aluno 2	12 (80%)
Aluno 3	15 (100%)
Aluno 4	15 (100%)
Aluno 5	7 (46,67%)
Média	12,8 (85,33%)

Na segunda fase, os alunos tiveram que avaliar e classificar 10 *bugs* selecionados aleatoriamente de cada uma dos sistemas, totalizando 50 *bugs*. Os resultados desta etapa estão apresentados na Tabela 5 a seguir:

Tabela 5 – Quantidade de concordância das avaliações externas com as classificações manuais prévias

	Apache	**Eclipse**	**Kernel**	**Mozilla**	**Open Office**	**TOTAL**	**%**
Aluno 1	9	8	10	7	8	42	84%
Aluno 2	10	7	9	8	8	42	84%
Aluno 3	8	9	8	9	8	43	86%
Aluno 4	9	6	9	8	8	40	80%
Aluno 5	7	7	7	4	8	33	66%
TOTAL	43	37	43	36	41	200	80%

Os resultados apresentados foram considerados satisfatórios, visto que apresentaram, em média, 80% de concordância com os resultados obtidos manualmente na etapa anterior.

Tabela 6 – Taxa de concordância por grupo de severidade

Grupo	Porcentagem
Severos	97%
Não Severos	77%
Normais	51%

Partindo do princípio que este fora o primeiro contato dos alunos com uma pesquisa científica e exclusivamente com os dados, pode-se avaliar como positivo o resultado final e como baixa a taxa de desacordo. Além disso, como pode ser visto na Tabela 6, a menor taxa de concordância aconteceu no grupo de *bugs* reclassificados manualmente como normais. Isso evidencia ainda mais o quão difícil e subjetivo é definir e classificar um *bug* com este nível de severidade e, percebe-se que, quando há uma necessidade de reclassificação, especialmente no caso de *bugs* severos, esta reclassificação é evidente (visto os 97% de concordância para este cenário). É importante ressaltar também que, apesar de ser a menor taxa de concordância, houve acerto em mais da metade dos *bugs* reclassificados com a severidade normal, e, destacando mais uma vez que foi a primeira vez que os alunos lidaram com estes dados, o resultado também pode ser considerado satisfatório.

Em seguida, na etapa oito, os *bugs* avaliados manualmente foram utilizados como base de treinamento para o classificador automático desenvolvido. Este classificador foi desenvolvido utilizando-se da ferramenta *open source Mallet*. Foram inseridos na ferramenta tanto os dados da base de treinamento citada, quanto os demais *bugs*, para serem analisados pelo classificador. Este utilizou três algoritmos distintos de classificação para aferir seus resultados (*Näive Bayes, Max Ent* e *Winnow*), que estão expostos a seguir, nas Tabelas 7, 8 e 9.

Diante dos resultados apresentados podemos verificar que, considerando todos os sistemas avaliados, em termos de quantidade, o melhor resultado foi o do algoritmo *Winnow*, que conseguiu reclassificar ao todo 35,75% (13,9% de severos e 21,85% de não severos). Porém, ao analisar cada um dos sistemas separadamente, com relação às quantidades, temos que: para o *Apache*, o algoritmo que obteve maior taxa de reclassificação foi o *Winnow*, com 75,3% dos *bugs* reclassificados (sendo 31,1% em severos e 44,2% em não severos); no caso do *Eclipse*, o melhor resultado ocorreu com o algoritmo *Max Ent*, onde 55,9% dos *bugs* foram reclassificados (29% como severos e 26,9% como não severos); já no *Kernel*, o melhor cenário também ocorreu na reclassificação feita pelo algoritmo *Max Ent*, com um total de 64,9% dos *bugs* reclassificados (com 31,3% severos e 33,6% não severos); o *Mozilla* obteve como melhor taxa de reclassificação a do algoritmo *Winnow*, onde 37,8% dos *bugs* foram reclassificados (14,2% como severos e 23,6% como não severos); e por fim, o *Open Office* obteve melhor resultado também com o *Winnow*, sendo 60,9% dos *bugs* reclassificados (31% severos e 29,9% não severos). Nota-se que o algoritmo *Näive Bayes*

não obteve bons resultados, pois tanto no geral, quanto em particular, ele não consegiu obter melhor classificação em nenhuma das situações.

Também é importante salientar que os resultados foram considerados satisfatórios, principamente por levarmos em consideração as taxas de acerto aferidas na etapa anterior a esta, onde houve validação externa. O resultados de 85,33% no primeiro momento, e 80% no segundo, trazem certa confiança para os resultados apresentados nesta etapa de classificação automática, que também foi pós-validada e seus resultados serão apresentados na próxima seção.

Tabela 7 – Resultados quantitativos da reclassificação do algoritmo Max Ent

Sistema	Severos	Não Severos	Normais
Apache	785 (42%)	595 (31,8%)	489 (26,2%)
Eclipse	**17487 (29%)**	**16222 (26,9%)**	26533 (44,1%)
Kernel	**513 (31,3%)**	**551 (33,6%)**	576 (35,1%)
Mozilla	19193 (8,6%)	10746 (4,8%)	192032 (86,6%)
Open Office	256 (19,9%)	403 (31,3%)	627 (48,8%)
TOTAL	38234 (13,3%)	28517 (9,9%)	220257 (76,8%)

Tabela 8 – Resultados quantitativos da reclassificação do algoritmo Näive Bayes

Sistema	Severos	Não Severos	Normais
Apache	44 (2,35%)	156 (8,3%)	1669 (89,35%)
Eclipse	3626 (6%)	6198 (10,3%)	50418 (83,7%)
Kernel	535 (32,6%)	223 (13,6%)	882 (53,8%)
Mozilla	20982 (9,45%)	28863 (13%)	172126 (77,55%)
Open Office	324 (25,2%)	343 (26,7%)	619 (48,1%)
TOTAL	25511 (8,9%)	35783 (12,5%)	225714 (78,6%)

Tabela 9 – Resultados quantitativos da reclassificação do algoritmo Winnow

Sistema	Severos	Não Severos	Normais
Apache	**582 (31,1%)**	**826 (44,2%)**	461 (24,7%)
Eclipse	6894 (11,4%)	8905 (14,8%)	44443 (73,8%)
Kernel	528 (32,2%)	220 (13,4%)	892 (54,4%)
Mozilla	**31528 (14,2%)**	**52381 (23,6%)**	138062 (62,2%)
Open Office	**399 (31%)**	**384 (29,9%)**	503 (39,1%)
TOTAL	**39931 (13,9%)**	**62716 (21,85%)**	184361 (64,25%)

Uma análise foi conduzida visando verificar qual a compatibilidade dentre as classificações automáticas dos três algoritmos. Assim sendo, procurou-se validar qual a

quantidade de *bugs* que foram classificados da mesma forma pelos três ou apenas em dois. Os resultados estão expostos a seguir, nas Tabelas 10 a 13.

Tabela 10 – Quantidade de Bugs que tiveram a mesma classificação nos três algoritmos

Sistema	Severos	Não Severos	Normais
Apache	41	88	179
Eclipse	1396	2458	20475
Kernel	202	88	360
Mozilla	5536	3881	113144
Open Office	123	152	249
TOTAL	7298	6667	134407

Tabela 11 – Quantidade de Bugs que tiveram a mesma classificação nos algoritmos Max Ent e Winnow

Sistema	Severos	Não Severos	Normais
Apache	356	363	180
Eclipse	3270	4012	20920
Kernel	225	115	368
Mozilla	6631	4128	121829
Open Office	129	178	274
TOTAL	10611	8796	142941

Tabela 12 – Quantidade de Bugs que tiveram a mesma classificação nos algoritmos Max Ent e Näive Bayes

Sistema	Severos	Não Severos	Normais
Apache	44	109	488
Eclipse	3142	4419	25126
Kernel	393	215	530
Mozilla	10208	6920	161099
Open Office	222	311	511
TOTAL	14009	11974	187754

Estes resultados nos indicam que existem dois cenários importantes na reclassificação de *bugs*. O primeiro deles é onde desejamos encontrar uma reclassificação de um *bug* com maior confiança. Para isto, temos os resultados encontrados de forma igual nos três algoritmos, ou seja, temos confirmação tripla para a reclassificação daquele *bug*. O segundo cenário acontece quando a necessidade da reclassificação é menor. Diante disso, a análise pode ser feita das seguintes formas:

1. Maior concordância nas classificações de *bugs* severos. Neste caso, a combinação se saiu melhor foi entre os algoritmos *Max Ent e Näive Bayes*;

Tabela 13 – Quantidade de Bugs que tiveram a mesma classificação nos algoritmos Näive Bayes e Winnow

Sistema	Severos	Não Severos	Normais
Apache	41	111	442
Eclipse	1693	3520	40498
Kernel	297	95	610
Mozilla	11199	19048	122210
Open Office	183	168	308
TOTAL	13413	22942	164068

2. Maior concordância nas classificações de *bugs* não severos. Neste caso, a combinação que se saiu melhor foi entre os algoritmos *Näive Bayes e Winnow*;

3. Maior concordância nas classificações de *bugs* normais. Neste caso, a combinação que se saiu melhor foi, novamente, entre os algoritmos *Max Ent e Näive Bayes*;

4. Maior concordância geral, entre as três categorias de severidades possíveis. Neste caso, a combinação que se saiu melhor foi, mais uma vez, entre os algoritmos *Max Ent e Näive Bayes*, pois tiveram ao todo 213737 *bugs* em acordo.

Sumário para Pergunta de Pesquisa 2: Todos os resultados apresentados nesta seção servem para confirmar a hipótese 2 deste trabalho, indicando que nem todos *bugs* classificados como normal refletem a real severidade aferida inicialmente ao mesmo. Neste aspecto, foi possível criar uma base de treinamento para o classificador automático, validada posteriormente por pessoas externas (com taxa de concordância de 80%). É importante ressaltar que esta validação externa ocorreu para observar principalmente se a metodologia proposta poderia ser compreendida e aplicada na prática por outras pessoas. A taxa obtida pode ser considerada satisfatória, pois todos participantes da etapa de validação externa foram instruídos adequadamente antes de realizarem suas respectivas tarefas. Com essa base, pode-se então aferir automaticamente a severidade de todos os *bugs* classificados como normais, e então indicar que, no melhor caso, 35,75% dos *bugs* foram reclassificados, o que representou mais de 100.000 *bugs* reclassificados. Além disso, a segunda pergunta de pesquisa é respondida, visto que foi possível realizar uma reclassificação automática dos *bugs* (seguindo a metodologia criada), e esta conseguiu encontrar uma quantidade satisfatória de *bugs* a serem reclassificados.

4.3 Pós validação e análises finais

Após a classificação automática, foi realizada uma pós validação dos resultados, visando mensurar sua precisão e dar início à última fase deste trabalho. Assim, foram

selecionados 600 *bugs* para conferência manual (seguindo mesmo método realizado na etapa que antecedeu a classificação automática), sendo 30 *bugs* para cada sistema (10 por nível de severidade severo, normal ou não severo), de acordo com as combinações de resultados dos algoritmos, apresentada anteriormente ou seja, foram escolhidos 10 *bugs* reclassificados como severos, 10 como não severos e 10 normais, nos três algoritmos de classificação, do sistema *Apache*, por exemplo. É importante ressaltar que os *bugs* analisados nesta fase foram retirados daqueles reclassificados de forma automática na fase anterior. Os resultados obtidos podem ser verificados a seguir, na Tabela 14:

Tabela 14 – Quantidade de avaliações iguais

Sistema	**Max Ent x Näive Bayes x Winnow**	**Max Ent x Winnow**	**Max Ent x Näive Bayes**	**Näive Bayes x Winnow**	**Total**
Apache	28 (93,33%)	26 (86,67%)	28 (93,33%)	27 (90%)	109 (90,83%)
Eclipse	23 (76,67%)	22 (73,33%)	21 (70%)	20 (66,67%)	86 (71,67%)
Kernel	25 (83,33%)	23 (76,67%)	26 (86,67%)	19 (63,33%)	93 (77,50%)
Mozilla	24 (80%)	22 (73,33%)	21 (70%)	23 (76,67%)	90 (75,00%)
Open Office	29 (96.67%)	27 (90%)	27 (90%)	22 (73,33%)	105 (87,50%)
Total	129 (86%)	120 (80%)	123 (82%)	111 (74%)	

De acordo com a tabela apresentada anteriormente, podemos identificar que o melhor caso ocorre nos *bugs* avaliados com a mesma severidade pelos três algoritmos. Nesta situação, tivemos que em 86% dos *bugs* avaliados manualmente na pós validação houve concordância com os resultados dos algoritmos automáticos. Com 82% de assertividade, a associação dos algoritmos *NäiveBayes* e *Max Ent* apresentou segunda melhor taxa de concordância, enquanto a associação dos algoritmos *Winnow e Max Ent* ficou em terceiro lugar, com 80% de acertos. Por fim, com 74% de conformidade, os algoritmos *Näive Bayes e Winnow* associados, obtiveram pior resultado em relação aos demais, porém também um resultado considerado satisfatório.

A análise de tais resultados nos permite indicar aos usuários que, caso necessitem de uma classificação automática mais precisa, a união dos três algoritmos utilizados é a mais indicada, pois trará resultados com maior acurácia. No entanto, não descarta as demais classificações, que também obtiveram resultados aceitáveis. Também é importante salientar que a tabela anterior apresentou as quantidades de concodâncias entre a pós validação e os resultados da classificação automática, devido a este fato, a soma dos totais não chega ao valor de 600 *bugs*, que foi a quantidade selecionada, e só poderia ser atingida, se todos os resultados fossem iguais, ou seja, se houvesse 100% de compatibilidade em todos os casos.

Uma das principais contribuições deste trabalho tem como foco auxiliar o desenvolvedor do *software*. É claro pensarmos que, em se tratando de correção de *bugs*, os

desenvolvedores mais experientes sejam alocados para correções mais complexas e difíceis, enquanto desenvolvedores iniciantes trabalhem no reparo de *bugs* mais simples. Considerando isto, e o fato já mostrado neste trabalho da desproporcionalidade encontrada quanto à *bugs* classificados inicialmente com a severidade Normal, analisou-se o número de comentários feitos nos *bugs* reclassificados pelo classificador automático em cada nível de severidade (severo, normal e não severo). Para isso, foram selecionados aleatoriamente 15 *bugs* de cada um dos sistemas aqui estudados, dentre aqueles validados manualmente após a classificação automática. Os resultados obtidos podem ser verificados a seguir.

Tabela 15 – Média de comentários com desvio padrão

Sistema	**Severos**	**Normais**	**Não Severos**
Apache	8,07 (8,00)	5,07 (5,00)	3,40 (2,00)
Eclipse	12,00 (10,30)	5,80 (7,70)	4,67 (2,40)
Kernel	18,00 (13,60)	14,87 (14,60)	7,07 (3,90)
Mozilla	38,73 (68,40)	8,07 (8,70)	5,53 (3,30)
Open Office	12,93 (3,20)	4,93 (1,80)	4,40 (1,35)
Média Geral	17,95 (32,60)	7,75 (9,50)	5,01 (2,95)

Os resultados demonstram que os *bugs* reclassificados como severos possuem uma média maior de comentários. Isso pode ser explicado pelo fato de que, como o *bug* inicialmente estava classificado como normal, ele não fora alocado de forma adequada para correção. Desta forma, o desenvolvedor que iniciou a correção do *bug* teve dificuldades e necessitou de auxílio de outra pessoa com conhecimento técnico maior para tentar finalizar a tarefa. É o caso de alguns dos *bugs* analisados, como por exemplo o *bug* número 306173, do *Eclipse*, que necessitou de intervenção de 13 pessoas para que sua correção fosse enfim implementada. Além disso, o último comentário, que fechou a correção do *bug*, não foi feito pelo desenvolvedor que havia selecionado ele inicialmente para correção. Outro fator importante de se destacar ocorreu no caso dos *bugs* reclassificados como não severos. Estes, por sua vez, tiveram uma média de comentários menor, devido ao fato de que eram casos com menor complexidade. Entretanto, é necessário observar que o desvio padrão em relação aos *bugs* severos é significativamente maior.

Visando entender esta influência da má classificação na atividade de correção, foram selecionados 15 *bugs* (sendo estes os top três *bugs* com mais comentários de cada um dos sistemas), do grupo de severos escolhidos anteriormente para pós-validação, com intuito de analisar o conteúdo dos comentários proferidos. Após analisar estes comentários, alguns fatores chamaram a atenção, e estão listados abaixo:

- Em 73,33% dos casos, o último comentário do *bug* não foi de quem o selecionou inicialmente;

- Em 73,33% casos houve cinco ou mais pessoas interagindo nos comentários em busca da correção do *bug* (sendo que em quatro destes casos, mais de 11 pessoas interagiram nos comentários do *bug*);
- Os comentários dos outros desenvolvedores eram em geral: resposta a dúvidas do desenvolvedor sobre como realizar a correção, questionamentos sobre mais detalhes do ocorrido (para que se pudesse auxiliar o desenvolvedor), anexos importantes para correção (*logs*, telas, etc), relacionamento com outro *bug* já corrigido ou com outro *bug* que estaria aguardando aquela correção;

Desta forma podemos indicar que o processo de correção destes *bugs* poderia ter sido mais rápido e eficiente se a classificação do mesmo fosse aferida de forma correta. Como isso não ocorreu, a correção começou a ser feita por um desenvolvedor que necessitou de auxílio de outros para finalizar a mesma ou até mesmo deixou para outra pessoa o finalizar, postergando o término daquela tarefa. Com isso, evidenciamos a hipótese 3 deste trabalho, visto que a má classificação das severidades teve impacto no trabalho de correção dos *bugs*, gerando maior discussão entre diferentes desenvolvedores, na busca pela correta efetivação do reparo necessário, o que também responde a terceira pergunta de pesquisa.

Sumário para Pergunta de Pesquisa 3: a má classificação de alguns *bugs*, que foram inicialmente classificados como normais, influencia diretamente na correção a ser realizada pela equipe de desenvolvimento, especialmente no caso onde o *bug* deveria ter sido classificado como severo. Para estes casos, percebeu-se em uma parte destes *bugs*, que o número de comentários e interações entre desenvolvedores distintos é maior, pois aquele desenvolvedor que iniciou a correção do mesmo teve dificuldade no processo e necessitou do auxílio dos demais. Desta forma, os esforços gastos na correção de *bugs* foi maior, visto que houve maior discussão sobre o mesmo, com diversas outras pessoas.

Capítulo 5

Trabalhos Relacionados

Nesta seção vamos apresentar trabalhos que principalmente analisaram *bugs* de sistemas. Em um primeiro momento será mostrado um trabalho que estuda o tempo de correção de *bugs* para um conjunto de dados. Em seguida serão apresentados trabalhos que identificaram má classificação de *bugs* em outros conjuntos de dados. Depois serão expostos os trabalhos que tentaram predizer informações relativas aos *bugs* por meio do estudo das características dos dados e, principalmente, de análise textual. Também serão apresentados em seguida, trabalhos que estudaram como é feita a triagem de *bugs* pelos desenvolvedores para correção, e o primeiro trabalho que procurou investigar os *bugs* normais de fato. Ao final, será apresentado como nosso trabalho pretendeu diferenciar-se dos demais e trazer novas contribuições.

Em um trabalho anterior, realizou-se um estudo empírico visando analisar o tempo gasto para efetivamente se corrigir *bugs*, ou seja, tempo transcorrido desde que o desenvolvedor seleciona o *bug* para correção, até a implementação da mesma no sistema (CANFORA et al., 2011). Segundo este estudo, que fora realizado utilizando dados de quatro projetos (*Eclipse, Mozilla, OpenLDAP e Vuze*), o tempo de sobrevivência de um *bug* depende principalmente de sua complexidade e de alguns fatores, como por exemplo a severidade do mesmo. Seu principal objetivo foi analisar a relação entre as construções e correções realizadas no código fonte do *software*, para se corrigir um bug, e a permanência destes problemas no sistema. Ao final, concluiu que existem determinadas construções de código que são altamente correlacionadas com *bugs*, e que estas construções merecem uma maior atenção por parte da equipe de desenvolvimento do software. Neste trabalho, vemos a importância da severidade e seu impacto na correção final do mesmo, que é um dos pontos de estudo do presente trabalho.

Por outro lado, em outro estudo, foi observado que uma parte dos *bugs* de seu conjunto de dados estava mal classificada (HERZIG; JUST; ZELLER, 2013). Por meio da análise de cerca de 7000 *bugs* extraídos das ferramentas *Jira* e *Bugzilla*, verificaram que mais de um terço deste conjunto de dados não consistia em um *bug* propriamente dito.

Esse é um ponto de atenção que tivemos em nosso conjunto de dados. Porém, há uma clara distorção entre as informações reportadas e o que desenvolvedor realmente necessita (ZIMMERMANN et al., 2010). Uma das causas deste fenômeno é a falta conhecimento pleno tanto da ferramenta de gestão de *bugs*, quanto do sistema em si, por parte do testador. Isso eleva o tempo de correção final e também pode gerar más classificações, como as encontradas em (HERZIG; JUST; ZELLER, 2013). Nosso trabalho diferenciou-se deste ao selecionar uma maior quantidade de *bugs* e trabalhar com um número maior de sistemas, podendo assim, expandir o cenário de observação.

Alguns estudos tiveram por objetivo tentar predizer fatores que os testadores devem classificar ao relatar um novo *bug*, de certa forma, visando minimizar a má classificação mencionada. Os dados do *Eclipse* e do *GNOME* foram utilizados para tentar predizer a severidade dos novos *bugs* por (LAMKANFI et al., 2011). Eles partiram de uma análise textual, com base na descrição dos bugs antigos, e classificaram bugs como sendo severos ou não severos. O problema deste estudo foi que eles deixaram de lado uma parcela de *bugs* classificados como normais, não levando em consideração dados que podem chegar a 85% de toda a amostra, como apresentado em nosso estudo. Por outro lado, em um estudo recente, múltiplos fatores foram considerados, em um algoritmo chamado DRONE, para predizer a prioridade de *bugs*, também do *Eclipse* (TIAN; LO; SUN, 2013). No entanto, neste estudo, eles obtiveram correlações menores, comparados aos obtidos pelo estudo anterior, enquanto neste a correlação ficou em 58,61%, no anterior tivemos algo em torno de 90%. Tentando diferenciar-se destes estudos, o presente trabalho leva em consideração como principal artefato os bugs classificados como normais, que foram relevados anteriormente.

Nosso estudo assemelha-se ao que foi realizado por (WEISS et al., 2007) e por (KIM et al., 2007). O primeiro caso apresenta uma abordagem semelhante ao primeiro estudo citado nesta seção, ao tentar prever esforço empregado pelos desenvolvedores para correção de *bugs* do projeto *JBoss* (tempo este que se inicia a partir da seleção do *bug* pelo desenvolvedor, e termina com sua implementação). Tal estudo baseou-se em técnicas de análise textual de *bugs* antigos, com intuito de prever o esforço que poderia ser gasto pela equipe de desenvolvimento para correção de novos problemas. No entanto, os próprios autores mencionam que os bugs possuem vários outros campos, que não foram levados em consideração, porém, podem auxiliar a melhorar essa previsão dos tempos de correção. Já no segundo caso, os autores tentaram predizer quais arquivos e/ou entidades dos sistemas analisados estavam mais propensas a falhar. Para isso, utilizaram de informações de bugs antigos visando detectar quais locais no software viriam a falhar futuramente. Desta forma, nosso trabalho também se utiliza de dados históricos para inicialmente criar um classificador automático que procura reclassificar os bugs inicialmente relatados como normais. No entanto, ele não se limita a isso, visto que pode ser utilizado para identificação da severidade de um futuro bug a ser relatado na ferramenta de gestão.

Porém, a má classificação dos *bugs* pode inclusive prejudicar esta tarefa de predição (KOCHHAR; LE; LO, 2014). Além disso, sabe-se que a atividade de triagem de *bugs* não é uma tarefa tão trivial, conforme o trabalho feito (ANVIK; HIEW; MURPHY, 2005). Neste trabalho, foi feita uma análise de *bugs* reportados na ferramenta de gestão *Bugzilla*, e conseguiram identificar um problema importante para realização da triagem de *bugs*: muitas pessoas têm acesso a ferramenta e podem fazer postagens indiscriminadamente. Desta forma, eles se depararam com *bugs* duplicados, irrelevantes e até mal classificados, fatores que impactam diretamente na correção que deve ser feita. A principal forma de má classificação encontrada foi no alto índice de *bugs* relatados como Normais. Procurando diferenciar-se dos demais estudos, nossa pesquisa aprofundou-se neste conjunto, para tentar entender melhor e reclassificá-lo.

Enfim há uma preocupação exclusiva com os *bugs* normais de um conjunto de *bugs* de sistemas que compõem a ferramenta *Eclipse,* feita por (Ripon K. Saha et al., 2015). Como hipóteses do estudo, foi sugerido que *bugs* classificados como normais não refletem a real severidade daquela inconsistência encontrada e que os relatores de *bugs* não se preocupam em alterar a severidade quando ela é classificada como padrão (normal). Diante disso, foi realizada uma seleção de 500 *bugs* aleatórios do conjunto trabalhado (que possui *bugs* reportados e corrigidos de 2006 a 2011). A partir desta análise, foi identificado que 65% dos *bugs* analisados não são Normais e que a causa da má classificação se dá pela subjetividade do campo. Nosso estudo assemelha-se a este, porém traz dados mais atuais e mais amplos, visto que trabalhamos com dados de cinco diferentes ferramentas.

Em nosso trabalho, analisamos os fatores que levam os *bugs* a serem relatados, principalmente, com a severidade normal. Estes fatores foram analisados de acordo com os comportamentos observados dos desenvolvedores e testadores perante todo o processo de correção, como por exemplo as descrições dos bugs relatados e a presença de termos que se destacam naqueles conjuntos considerados como severos e não severos. Como todos os dados estudados no presente trabalho foram relatados na ferramenta Bugzilla, alguns fatores chamaram a atenção para nosso estudo, podemos citar como exemplos: a severidade, o componente, o produto e o sumário descritivo do *bug*. Estes dados podem trazer informações como a urgência de um *bug*, quais partes dos sistemas são mais defeituosas e quais palavras são recorrentes em determinados grupos.

É importante ressaltar que todos os trabalhos apresentados escolheram uma determinada parte relacionada aos *bugs* para explorar e que, quando realizaram alguma predição, não levaram em consideração o impacto trazido pela má classificação de alguns *bugs*. Em nosso estudo, temos como objetivo relacionar algumas características antes estudadas de forma separada (sumário, severidade, produtos e componentes), em conjunto. Além disso, sabemos que foi importante realizar um refinamento profundo, com intenção de selecionar apenas *bugs* registrados como solucionados e implementados, evitando distorções

nos resultados que serão apresentados ao final deste trabalho.

Capítulo 6

Considerações Finais

Este trabalho apresentou um estudo que permitiu avaliar uma porção dos dados ainda pouco estudada anteriormente (*bugs* classificados inicialmente com a severidade normal), de um conjunto de *bugs* extraídos do repositório *Bugzilla*, de cinco sistemas amplamente utilizados pela comunidade: *Apache, Eclipse, Kernel, Mozilla* e *Open Office*. Os resultados obtidos neste trabalho enfatizam a importância de se ter cuidado ao relatar novos *bugs*, face ao alto número de má classificação encontrada. Na amostra utilizada neste trabalho, cerca de 85% do total analisado foi classificado com a severidade normal, sendo que nossos resultados apontaram que desta porção, entre 21% e 36% deveriam ter sido relatados de outra forma (o que representa entre 70.000 e 130.000 *bugs*). Além disso mostramos casos onde foram identificados *bugs* que, após terem sido mal classificados, foram alocados inadequadamente para correção, visto que o desenvolvedor necessitou de auxílio da comunidade para finalizar a correção, atrasando a correta implementação do reparo de problemas mais complexos.

6.1 Limitações da pesquisa

Neste trabalho, algumas limitações podem ser apontadas. Em primeiro lugar, temos os sistemas que foram fonte de dados para esta pesquisa. Tentou-se buscar um número variado de aplicações, que são utilizados em diferentes contextos, porém, foram utilizadas informações apenas de sistemas *open source* e provindos dos repositórios do *Bugzilla*. O estudo de aplicações privadas e coletadas de outros repositórios poderia enriquecer trabalhos futuros. Além disso, foram estudados sistemas que operam apenas em computadores pessoais ou servidores. O estudo de aplicações móveis também pode aumentar a abrangência do estudo.

Em segundo lugar, apenas algumas das características dos *bugs* foram levadas em consideração para definição da metodologia de reclassificação dos *bugs*. Não é claro que o acréscimo de outras características melhoraria os resultados aqui obtidos, porém tal análise

poderá ser realizada futuramente e trazer novas informações. Alguns exemplos destas outras características seriam: o Sistema Operacional, a versão do sistema e os anexos de cada *bug*.

Por último, o uso de outros algoritmos de classificação também poderia ser empregado. Devido à limitação da ferramenta *Mallet* (utilizada para realizar as reclassificações), apenas três algoritmos foram utilizados. O estudo de outras ferramentas pode auxiliar na tentativa de aplicação de outros algoritmos. Também é importante ressaltar que este trabalho resultou em duas submissões para conferências internacionais, porém, estas submissões foram rejeitadas. No entanto, mesmo com a publicação não efetivada, os comentários proferidos pelos examinadores auxiliaram este trabalho a evoluir.

6.2 Trabalhos Futuros

O estudo realizado neste trabalho pode ser expandido para obtenção de novos resultados. Como mencionado na seção de limitações da pesquisa, o uso de conjunto de dados provindos de outros repositórios e de sistemas privados, pode ampliar os resultados de reclassificação de *bugs* e indicar se o problema da má classificação é uma situação particular dos cenários estudados neste trabalho, ou podem sem ampliados para outras situações. Dados como os do sistema operacional *Windows,* utilizados por (GUO et al., 2010), e da *NASA*, no estudo de (MENZIES; MARCUS, 2008), são um bom exemplo para serem avaliados no contexto deste trabalho e comparados com os resultados obtidos para os sistemas *open source*. Com isso, além de identificar se o problema investigado é uma situação particular ou geral, também podem ser avaliadas outras características presentes nos repositórios dos sistemas privados, que poderiam auxiliar na correção dos *bugs* de sistemas *open source*, e vice-versa.

Também é recomendado que novos algoritmos de classificação sejam utilizados. A comparação com eles pode ratificar os resultados obtidos ou ainda apresentar situações novas a serem analisadas. Desta forma, outros estudos já realizados podem ser também revistos, visando avaliar se seus resultados são alterados após a inclusão dos *bugs* reclassificados, visto que desconsideraram os *bugs* classificados como normais em suas pesquisas.

Finalmente, a análise de outras características dos *bugs* pode ser feita e seu impacto nos resultados obtidos podem trazer informações que levem a novas contribuições e melhorias no estudo realizado. A inclusão de sistemas provindos de outros repositórios e/ou de sistemas desenvolvidos por empresas privadas, pode auxiliar neste aumento de características.

Referências

ANVIK, J.; HIEW, L.; MURPHY, G. C. Coping with an Open Bug Repository. In: *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*. New York, NY, USA: ACM, 2005. p. 35 – 39.

ANVIK, J.; HIEW, L.; MURPHY, G. C. Who Should Fix This Bug? In: *Proceedings of the 28th International Conference on Software Engineering*. New York, NY, USA: ACM, 2006. (ICSE '06), p. 361 – 370.

Apache Software Foundation. *Página oficial do conjunto de dados da ferramenta Apache - https://bugs.apache.org/*. 2015. Acesso em Janeiro de 2015. Disponível em: <https://bugs.apache.org/>.

Apache Software Open Office Foundation. *Página oficial do conjunto de dados da ferramenta Open Office - https://bz.apache.org/ooo/*. 2015. Acesso em Janeiro de 2015. Disponível em: <https://bz.apache.org/ooo/>.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, v. 1, n. 1, p. 11 – 33, Jan 2004.

BARNSON, M. P. *The Bugzilla Guide*. 2006. Acesso em Julho de 2015. Disponível em: <https://www.bugzilla.org/docs/2.16/html/>.

BETTENBURG, N. et al. Quality of Bug Reports in Eclipse. In: *Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology eXchange*. New York, NY, USA: ACM, 2007. p. 21 – 25.

BORTIS, G.; HOEK, A. van der. PorchLight: A Tag-based Approach to Bug Triaging. In: *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013. (ICSE '13), p. 342 – 351.

CANFORA, G. et al. How Long Does a Bug Survive? An Empirical Study. In: *18th Working Conference on Reverse Engineering (WCRE)*. [S.l.: s.n.], 2011. p. 191 – 200.

CHATURVEDI, K.; SINGH, V. Determining Bug severity using machine learning techniques. In: *CSI Sixth International Conference on Software Engineering (CONSEG)*. [S.l.: s.n.], 2012. p. 1 – 6.

Eclipse Foundation. *Página oficial do conjunto de dados da ferramenta Eclipse - https://bugs.eclipse.org/bugs/*. 2015. Acesso em Janeiro de 2015. Disponível em: <https://bugs.eclipse.org/bugs/>.

Eclipse Foundation. *TPTP Development Process*. Ottawa, Ontario, Canada: [s.n.], 2015. Acesso em Julho de 2015. Disponível em: <http://www.eclipse.org/tptp/home/documents/process/development/bugzilla.html>.

GARCIA, H. V.; SHIHAB, E. Characterizing and Predicting Blocking Bugs in Open Source Projects. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2014. (MSR 2014), p. 72 – 81.

GOLDING, A. R.; ROTH, D. A Winnow-Based Approach to Context-Sensitive Spelling Correction. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 34, n. 1-3, p. 107 – 130, February 1999.

GRAHAM, S.; WEINGART, S.; MILLIGAN, I. *Getting Started with Topic Modeling and MALLET*. Carleton University: [s.n.], 2012. Acesso em Julho de 2015. Disponível em: <http://programminghistorian.org/lessons/topic-modeling-and-mallet>.

GUO, P. J. et al. Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows. In: *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*. New York, NY, USA: ACM, 2010. (ICSE '10), p. 495 – 504.

HERZIG, K.; JUST, S.; ZELLER, A. It's Not a Bug, Its a Feature: How Misclassification Impacts Bug Prediction. In: *Proceedings of the 2013 International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2013. (ICSE '13), p. 392 – 401.

HUANG, C.-Y.; KUO, C.-S.; LUAN, S.-P. Evaluation and Application of Bounded Generalized Pareto Analysis to Fault Distributions in Open Source Software. *IEEE Transactions on Reliability*, v. 63, n. 1, p. 309 – 319, March 2014.

IQBAL, M.; RIZWAN, M. Application of 80/20 rule in software engineering Waterfall Model. In: *International Conference on Information and Communication Technologies - ICICT '09.* [S.l.: s.n.], 2009. p. 223 – 228.

KIM, S. et al. Predicting Faults from Cached History. In: *29th International Conference on Software Engineering, 2007. ICSE 2007.* [S.l.: s.n.], 2007. p. 489 – 498.

KOCHHAR, P. S.; LE, T.-D. B.; LO, D. It's Not a Bug, It's a Feature: Does Misclassification Affect Bug Localization? In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2014. (MSR 2014), p. 296 – 299.

KUO, C.-S.; HUANG, C.-Y. A study of applying the bounded Generalized Pareto distribution to the analysis of software fault distribution. In: *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*. [S.l.: s.n.], 2010. p. 611 – 615.

LAMKANFI, A. et al. Predicting the severity of a reported bug. In: *7th IEEE Working Conference onMining Software Repositories (MSR)*. [S.l.: s.n.], 2010. p. 1 – 10.

LAMKANFI, A. et al. Comparing Mining Algorithms for Predicting the Severity of a Reported Bug. In: *15th European Conference on Software Maintenance and Reengineering (CSMR)*. [S.l.: s.n.], 2011. p. 249 – 258.

Linux Kernel Organization. *Página oficial do conjunto de dados da ferramenta Kernel - https://bugzilla.kernel.org/*. 2015. Acesso em Janeiro de 2015. Disponível em: <https://bugzilla.kernel.org/>.

LYU, M. R. *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

MCCALLUM, A. *Página oficial da ferramenta Mallet - http://mallet.cs.umass.edu/index.php*. 2015. Acesso em Julho de 2015. Disponível em: <http://mallet.cs.umass.edu/index.php>.

MCCALLUM, A.; FREITAG, D.; PEREIRA, F. C. N. Maximum Entropy Markov Models for Information Extraction and Segmentation. In: *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. (ICML '00), p. 591 – 598.

MCCALLUM, A.; NIGAM, K. *A comparison of event models for Naive Bayes text classification*. 1998.

MENZIES, T.; MARCUS, A. Automated severity assessment of software defect reports. In: *IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2008. p. 346 – 355.

Mozilla Foundation. *Página oficial do conjunto de dados da ferramenta Mozilla - https://bugzilla.mozilla.org/*. 2015. Acesso em Janeiro de 2015. Disponível em: <https://bugzilla.mozilla.org/>.

MYERS, G. J.; SANDLER, C. *The Art of Software Testing*. [S.l.]: John Wiley & Sons, 2004.

PIGOSKI, T. M. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. New York, NY, USA: John Wiley & Sons, 1996.

RIOS, E. et al. *Base de Conhecimento em Teste de Software*. 3.ed. ed. São Paulo: Martins Fontes, 2012.

SERRANO, N.; CIORDIA, I. Bugzilla, ITracker, and other bug trackers. *IEEE Software*, v. 22, n. 2, p. 11 – 13, March 2005.

SHARMA, M. et al. Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: *12th International Conference onIntelligent Systems Design and Applications (ISDA)*. [S.l.: s.n.], 2012. p. 539 – 545.

SINGH, Y.; KAUR, A.; MALHOTRA, R. Empirical Validation of Object-oriented Metrics for Predicting Fault Proneness Models. *Software Quality Journal*, Kluwer Academic Publishers, Hingham, MA, USA, v. 18, n. 1, p. 3 – 35, March 2010.

TIAN, Y.; LO, D.; SUN, C. Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction. In: *19th Working Conference on Reverse Engineering (WCRE)*. [S.l.: s.n.], 2012. p. 215 – 224.

TIAN, Y.; LO, D.; SUN, C. DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis. In: *29th IEEE International Conference on Software Maintenance (ICSM)*. [S.l.: s.n.], 2013. p. 200 – 209.

WEISS, C. et al. How Long Will It Take to Fix This Bug? In: *Proceedings of the Fourth International Workshop on Mining Software Repositories* . Washington, DC, USA: IEEE Computer Society, 2007. (MSR '07).

XUAN, J. et al. Developer Prioritization in Bug Repositories. In: *Proceedings of the 34th International Conference on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2012. (ICSE '12), p. 25 – 35.

ZHANG, L. *Maximum Entropy Modeling*. University of Edinburgh: [s.n.], 2015. Acesso em Julho de 2015. Disponível em: <http://homepages.inf.ed.ac.uk/lzhang10/maxent.html>.

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

Apêndices

APÊNDICE **A**

Lista de Principais Produtos e Componentes dos Sistemas Analisados

APACHE:

- Componentes: *All, APR, APR-util, Catalina, Cluster, Connectors, Core, Core tasks, Documentation, fonts, general, HSLF, HSSF, HTTP, HWPF, isapi, Jasper, jdbc-pool, Library, Main, Manager, mod_jk, mod_proxy, mod_rivet, mod_ssl, Optional Tasks, Other, Packaging, pdf, POI Overall, POIFS, Servlet & JSP API, Standard Taglib, SXSSF, XSSF e XWPF;*
- Produtos: *Ant, Apache httpd-2, JMeter, POI, Tomcat 6, Tomcat 7;*

ECLIPSE:

- Componentes: *Agent, aggregator, Ant, API Tools, Architecture Council, Bugzilla, Build, bundles, cdo.core, cdo.db, cdt-build, cdt-build-managed, cdt-codan, cdt-core, cdt-debug, cdt-debug-dsf, cdt-debug-dsf-gdb, cdt-debug-edc, cdt-editor, cdt-indexer, cdt-parser, cdt-refactoring, Chart, cliente, Clientcore, Code Assist, Common, Compare, Compendium, Compiler, Components, Connectors, Core, Cross-Project, CVS, Data, DBWS , Debug, Debugger, Deployment, Diagram, Diagram Editor, Diagrams, Doc, Documentation, ecf.providers, ecf.releng, ecf.remoteservices, eclipse-build, Editor, EEF, EMF Forms, engine, Forms, Forums and Newsgroups, Foundation, Framework, General, Generators, Gerrit, Gerrit Connector, Git, GUI, Hudson, Hudson sandbox, IDE, IDE Core, IDE UI, Incubator, Infrastructure, Install, JavaGen, JavaScript-Gen, JAXB, JGit, JPA, JS Tools, JSF Tools, jst.j2ee, jst.jsp, jst.server, jst.ws, jst.ws.jaxws, Language, Launcher, LDP, LTTng, LuaDevelopmentTools, MailingLists,*

Main, Marketplace, modeling, MOFModel, MOXy, osgi, OTDT, other, Others, OTJ, p2, PlanetEclipse.org, Platform, Platforms, Plugins, Project, Project Management & Portal, Proposals and Reviews, ProR, RC, RDT, RDT.sync, releng, Report Designer, Report Engine, Resources, RIO, RM, SER, Runtime, RWT, Samples, Scout, Scout, SDK, SDO, server, Servers, Setup, Snipmatch, spdy, SWT, Systemtap, Table, Target, Targets, Team, Technologies, Teneo, TestSuite, Texo, Text, tooling, Tools, UI, unknown, User Assistance, UserInterface Views, web-admin, Website, websocket, Widgets, Wikitext, wizard, Workbench, wst.common, wst.html, wst.server, wst.sse, wst.xml, Xcore, Xpand e Xtext

- Produtos: *BIRT, BPMN2Modeler, CDT, Community, Dali JPA Tools, ECF, EclipseLink, ECP, EDT, EGit, EMF, EMFCompare, EMFT.facet, Equinox, JDT, Jetty, JSDT, JPA, Linux Tools, Lyo, MDT.MoDisco, Mylyn Reviews R4E, OCL, Orion, Papyrus, PDE, PDT, Platform, PTP, RAP, Recommenders, RTSC, Sapphire, Scout, Subversive, Target Management, TCF, TMF, Virgo, WTP Java EE Tools, WTP Source Editing e Xtend;*

KERNEL:

- Componentes: *BlockLayer, Bluetooth, btrfs, ext4, Hibernation/Suspend, i386, InputDevices, IPV4, kvm, man-pages, Network, network-wireless, NFS, Other, PCI, Platform_x86, Power-Video, SerialATA, Sound(ALSA), Staging, USB, Video(DRI-Intel), Video(DRI-nonIntel), Video(Other), Wireless e x86-64;*
- Produtos: *ACPI, Drivers, FileSystem e Networking;*

MOZILLA:

- Componentes: *Account Manager, Account Requests, Add-on Builder, Add-on Validation, Add-ons Manager, Admin/Editor Tools, Administration, affiliates.mozilla.org, Air Mozilla, Android Sync, API, Application Update, Awesomescreen, Backend, Bedrock, Blocklisting, Bluetooth, Bookmarks & History, Breakpad Integration, Bugzilla-General, Build Config, Buildduty Canvas: 2D, Canvas: WebGL, Client, Client: Desktop, Code Quality, Collections, Compatibility Tools, Consumer Pages, Conversation, Copy, Creating/Changing Bugs, cs / Czech, CSS Parsing and Computation, Database, Database Operations, DCOps, de / German, Design, Developer Pages, Developer Tools, Developer Tools: Console, Developer Tools: Debugger, Developer Tools: Inspector, Developer Tools: Performance Tools (Profiler/Timeline), Developer Tools: WebIDE, Disability Access APIs, Discussion Forums, Document Navigation, Documentation, DOM, DOM: Apps, DOM: Core & HTML, DOM:*

Device Interfaces, DOM: Events, DOM: IndexedDB, DOM: Workers, Download Manager, DXR, Editing, Editor, Eideticker, Elmo, English US, Events, Firefox Flicks, Firefox Sync: Backend, Firefox Sync: UI, Front-end, Gaia, Gaia::Browser, Gaia::Build, Gaia::Calendar, Gaia::Camera, Gaia::Clock, Gaia::Contacts, Gaia::Cost Control, Gaia::Dialer, Gaia::E-Mail, Gaia::Everything.me, Gaia::First Time Experience, Gaia::Gallery, Gaia::Homescreen, Gaia::Keyboard, Gaia::Loop, Gaia::Music, Gaia::Settings, Gaia::SMS, Gaia::System, Gaia::System::Lockscreen, Gaia System Window Mgmt, Gaia::UI Tests, Garbage Collection (mmGC), Gecko Profiler, General, General Automation, Geolocation, GonkIntegration, Graphics, Graphics, Panning and Zooming, Graphics: Layers, Graphics:Text, HTML: Parser, ImageLib, Infra, Infrastructure, Infrastructure: Other, Installer, Instant Messaging, Internationalization, IPC, JavaScript Engine, JavaScript Engine: JIT, Knowledge Base Software, L10N, Layout, Layout: Form Controls, Layout: Text, Libraries, Loan Requests, Localization, Location Bar, Login, Mail Window Front End, MakeAPI, Marionette, Menus Mercurial: hg.mozilla.org, Metro Operations, MFBT, Mobile, MOC: Incidents, Mochitest, Mozbase, Mozharness, Mozmill, Mozmill Tests, MozTrap, NetOps, NetOps: DC ACL Request, Networking, Networking: Cache, Networking: HTTP, NFC, opentochoice.org, Operations, Operations: Deployment Requests, Operations: Marketplace, OrangeFactor, OS.File, Other, Pages & Content, Panning and Zooming, Panorama, Payments/Refunds, PDF Viewer, Phonebook, Places, planet.mozilla.org, Platform Support, Plug-ins, plugins.mozilla.org, Popcorn Maker, Preferences, Preinstalled B2G Apps, Private Browsing, Public Pages, Query/Bug List, Questions, Release Automation, Release Engineering, Releases, RelOps, RelOps: Puppet, Repository Account Requests, reps.mozilla.org, Reviewer Tools, RIL, Search, Security, Security Assurance: Review Request, Security: PSM, Server, Server Operations, Server Operations: MOC, Server: Sync, Servicedesk, Session Restore, SocialAPI, sr / Serbian, Statistics, Storage, SVG, Tabbed Browser, Talos, TaskCluster, TBPL, Telemetry, Testing Infrastructure, Theme, Theme and Visual Design, Themes, Thimble, Thunderbird, Toolbars and Customization, Tools, Treeherder, User Interface, Users and Groups, Video/Audio, Virtual Machine, Virtualization, Web Analytics, Web Apps, Web Audio, Webapp, webmaker.org, WebOps: IT-Managed Tools, WebOps: Labs, WebOps: Other, WebOps: Product Delivery, WebRTC, WebRTC: Audio/Video, WebRTC: Networking, WebRTC: Signaling, Website, Widget, Widget: Android, Widget: Cocoa, Widget: Gtk, Widget: Win32, Wiki pages, www.drumbeat.org, www.seamonkey-project.org, XPCOM, XPConnect e XUL;

- Produtos: *addons.mozilla.org, bugzilla.mozilla.org, Core, Firefox, Firefox for Android, Firefox OS, Infrastructure & Operations, Marketplace, Mozilla Developer Network, Mozilla Localizations, Mozilla Services, mozilla.org, mozilla.org Graveyard, Release Engineering, Socorro, support.mozilla.org, Testing, Toolkit, Webmaker, Websites e*

www.mozilla.org;

OPEN OFFICE:

- Componentes: *api, chart, checking, code, editing, external prerequisites, formatting, general, help, issues, open-import, programming, printing, save-export, scripting, solenv, spell, testscripts, viewing, ui, www*
- Produtos: *Build Tools, Calc, Draw, General, Impress, Infrasctructure, QA, Writer;*

APÊNDICE **B**

Lista de Termos mais recorrentes em Bugs Severos e Não Severos de cada sistema

APACHE

- Severos: *request, exception, thread, work, fails, org, response, files, websocket, proxy, apr.*
- Não severos: *documentation, html, typo, wrong, missing, default, test, log, code, resource, set.*

ECLIPSE

- Severos: *npe, diagram, model, fails, work, doesn, broken, files.*
- *Não severos: page, remove, test, wrong, ui, dialog, message, add.*

KERNEL

- Severos: *bug, system, intel, panic, regression, usb, radeon, null, oops, unable, crash, driver, bisected, device, resume.*
- Não severos: *wrong, doesn, process, brightness, iwlwifi, test, pointless, pci, acpi, bss, acer, ath, warning, reports, wireless.*

MOZILLA

- Severos: *crash, assertion, failure, build, org, broken, browser, work, fails, test, mail, can't, need, using, java, missing, causes, release, please, trunk, server, cannot.*

- Não severos: r*emove, new, use, menu, button, dialog, warning, link, does, fix, wrong, code, bar, Should, text, after.*

OPEN OFFICE

- Severos: *aoo, symphony, crashes, open, table, update, Windows, writer, ip, save, undo, sidebar.*
- Não severos: *automation, ooo, bas, wrong, translation, dialog, build, distribute, cws, rc, final, work.*