Avaliação empírica de termos técnicos em *issues* de projetos opensource

Joselito Mota Júnior, Ivan Machado

[1]Departamento de Ciência da Computação, Universidade Federal da Bahia (UFBA), Salvador – BA – Brasil

{joselito.mota,ivan.machado}@ufba.br

Abstract. *The growing demand for robust and reliable software systems has led the scientific community and the software industry to investigate aspects related to defects reported in bug tracking systems. The overall goal is to understand the causes of defects, and thus establish strategies to prevent their occurrence and spread, reducing possible damage. Defects reported by users via issues in bug trackers reveal important information for developers. The present investigation carried out an analysis of data from 510,212 issues from 85 repositories of open source projects, available on Github. With the help of a list of 608 technical terms selected manually, the frequency in the use of terms by contributors to these repositories was analyzed. The study found the use of terms in the different fields of an issue, as well as the possibility of identifying the terms most used in these repositories. The results of this study point to potential gains in understanding the failure environment by presenting terms that are indicative of defects in fundamental repository structures.*

Resumo. *A demanda crescente por sistemas de software robustos e confiáveis tem levado a comunidade científica e a indústria de software a investigar aspectos relacionados aos defeitos reportados em sistemas de rastreamento de defeitos. O objetivo comum é compreender as causas dos defeitos, e assim estabelecer estratégias para prevenir sua ocorrência e propagação, reduzindo possíveis danos. Os defeitos reportados por usuários através de issues em rastreadores de defeitos revelam informações importantes para os desenvolvedores. A presente investigação realizou uma análise de dados provenientes de 510.212 issues de 85 repositórios de projetos de código-fonte aberto, disponíveis no Github. Com o auxílio de uma lista de 608 termos técnicos selecionados manualmente, foi analisada a frequência na utilização dos termos pelos contribuidores desses repositórios. O estudo encontrou o emprego dos termos nos diferentes campos de uma issue, bem como a possibilidade de identificação dos termos mais utilizados nestes repositórios. Os resultados deste estudo apontam para ganhos potenciais na compreensão do ambiente de falhas por apresentar termos que formam indicativos de defeitos em estruturas fundamentais do repositório.*

1. Introdução

A busca por soluções confiáveis é o grande motivo para que a pesquisa sobre métodos, processos e ferramentas de software seja guiado para encontrar soluções eficientes. Para isso, é necessário empregar estratégias de verificação e validação ao longo do ciclo

de vida de desenvolvimento do software. Entretanto, é fundamental antecipar-se aos possíveis defeitos decorrentes de erros de programação ou equívocos cometidos durante a especificação, modelagem e construção do software. Neste sentido, o estudo dos defeitos pode ser uma estratégia para prevenção de tais situações. Dados sobre as classificações podem ser aplicados a diversas finalidades, incluindo a análise causal de defeitos, o gerenciamento de projetos e a melhoria de processos de software [Zubrow 2009]. O estudo dos defeitos reportados por usuários pode trazer informações críticas para direcionar os esforços da equipe de desenvolvimento.

Então, torna-se necessário uma inspeção do ambiente de falha para compreensão dos erros encontrados no software [Sullivan and Chillarege 1991]. Assim é reforçado a importância da interpretação das mensagens contidas nos relatórios de erros fornecidos pelos usuários nas *issues*, para investigar todos fatores da falha. Estas informações podem trazer resultados relevantes para encontar o tipo de defeito, quais deles são mais reportados, atribuir uma *issue* para certo grupo de programadores e outras informações que podem auxiliar na administração do repositório e prevenir futuros defeitos. Com isso, o propósito deste estudo exploratório é analisar a atividade dos repositórios pelo uso dos termos utilizado por contribuidores dos projetos nas discussões das *issues*. Para isso, foram elaboradas as questões de pesquisa a seguir para a condução desse estudo:

QP1) Ao definir um catálogo de termos técnicos utilizados em Computação, é possível encontrar uma quantidade significativa desses termos em *issues* reportadas pelos usuários dos repositórios em um rastreador de defeitos *opensource* de grande utilização? O objetivo é encontrar linhas de sintetização para encontrar os termos mais comentados em *issues* dos repositórios. Se há *issues* reportadas, espera-se que nelas estejam reportados defeitos sobre as mais diversas áreas da solução. Investigando termos relacionados às áreas afins, esperamos encontrar uma tendência no foco do assunto em determinada deficiência do repositório.

QP2) Em quais elementos das *issues* dos repositórios são encontrados esses termos? O objetivo é identificar os pontos em que há a discussão dos defeitos. Se existe a tendência de haver termos mais específicos em descrições, ou se nos comentários são abordados termos para identificação do defeito, e quais são estes termos.

Deste modo, o estudo pretende contribuir na área de detecção e triagem de defeitos a partir da investigação dos defeitos reportados pelas *issues*, utilizando para tanto a relevância dos termos técnicos pela quantidade de ocorrências encontradas na base de dados. A frequência na utilização destes termos revela o quão relevante é determinado assunto na comunidade de desenvolvimento de software *opensource*.

O estudo analisou dados de 85 repositórios listados no desafio *MSR 2014 Mining Challenge Dataset*[1] com o total de 510.212 *issues* reportadas na plataforma Github e minerados utilizando um script de mineração desenvolvido para prover suporte à automatização das atividades de busca, limpeza e análise de dados. Após os pré-processamentos para ajustar a base de dados de textos do título, descrição e comentários das *issues* e a construção com pesquisa e validação utilizando dicionários online de termos específicos da área de computação e desenvolvimento de *software*, este estudo analisou a frequência desses termos técnicos nos repositórios. Assim, foram utilizados algoritmos

[1] https://ghtorrent.org/msr14.html

para contagem de frequência dos termos nas *issues*. Ao final foi realizada uma discussão sobre os resultados encontrados.

Além da análise das *issues* pela frequência dos termos reportados pelo usuários, realizamos a análise de sentimentos das *issues* [Boechat et al. 2019]. Esse estudo, que contou com a participação do aluno Joselito Júnior, foi premiado como melhor artigo do *VII Workshop on Software Visualization, Evolution and Maintenance* (VEM 2019)[2].

Ressaltamos que a concepção do presente estudo, bem como a implementação dos artefatos utilizados na avaliação empírica foram realizadas pelo aluno Joselito Júnior sob a orientação do Professor Ivan Machado.

2. Fundamentação Teórica

O uso cada vez maior de sistemas intensivos em software, por todas as áreas do conhecimento, tem levado a uma crescente demanda por melhorias na confiabilidade e na qualidade do software produzido [Sullivan and Chillarege 1991]. No sentido de que os defeitos em sistemas de software podem causar prejuízos dos mais diversos, utilizar-se de técnicas que possam antecipar-se a tais ocorrências, e evitar a sua propagação, tem sido fundamental no processo de desenvolvimento de software.

Em um primeiro momento, é importante compreender os tipos de defeitos que mais afetam sistemas de software, em domínios distintos. Neste sentido, é fundamental a construção de modelos-padrão para classificar tais defeitos e assim, possibilitar uma visão mais holística dos defeitos produzidos ao longo do processo de desenvolvimento de software [Zubrow 2009]. Por isso, a procura por métodos de documentação, avaliação e resolução de defeitos determinou comportamentos que culminaram na criação de ferramentas para reunir informações sobre defeitos em *softwares*. Pois, segundo [Sullivan and Chillarege 1991] um modelo claro de erros de software é fundamental para fazer testes de maneira eficaz e validar novos projetos.

Há uma relação direta entre problema e falha de software, sendo que um problema é ocasionado por uma ou mais falhas. As falhas são consideradas um subtipo de defeito e são classificadas usando os mesmos atributos como defeitos [Zubrow 2009]. Um defeito é uma imperfeição ou deficiência no produto que não segue os seus requisitos ou especificações que precisam ser reparadas [Zubrow 2009]. Estes podem ser reportados sob a forma de *issues* em rastreadores de defeitos de software. Segundo [Karahroudy and Tabrizi 2011], um defeito é uma propriedade do software que apresenta uma divergência próxima total ou local da especificação correta. Para [Card 1998], os erros relacionandos a dados e estrutura da dados são comuns em soluções de software. Ainda para [Grady 1996], um problema de codificação pode ser encontrado quando a lógica proposta não está correta. De posse destas informações temos que tipos específicos de defeitos podem surgir em determinados domínios de sistemas. Ao estudar termos específicos de áreas no desenvolvimento, é possível extrair informações relevantes para focar na resolução de defeitos.

3. Metodologia do trabalho

A busca por repositórios validados pela comunidade científica foi o primeiro passo executado nesta investigação. A Conferência *Mining Software Repositories (MSR)* realizada em

[2] `https://vem2019.github.io/`

2014 listou 90 repositórios para realização do desafio *MSR Challenge*. Considerando a MSR como uma das mais importantes conferências da área de Engenharia de Software, tal lista de repositórios foi utilizada como referência para este trabalho. Ao validar a disponibilidade destes repositórios, observamos que cinco deles não estavam disponíveis, o que restou em um conjunto final de 85 repositórios para análise. Ao término da mineração e posterior pré-processamento de dados, um estudo exploratório da base de dados apresentou 510.212 *issues* com 3.075.304 textos dos títulos, descrição e comentários, compondo 60.991.362 palavras descritas nas *issues*.

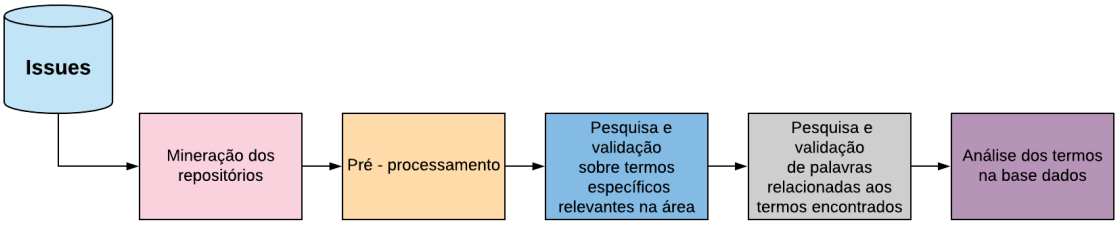

Figura 1. Visão geral do fluxo de atividades desenvolvidas.

As etapas desenvolvidas neste projeto são apresentadas no fluxo de trabalho da Figura 1. Para cada etapa são discutidas a seguir os procedimentos realizados para obtenção dos resultados.

3.1. Mineração dos repositórios

Para facilitar o processo de mineração, implementamos um *script* de mineração de dados em Python[3], utilizando a biblioteca PyGithub[4]. Este script serviu para extrair e armazenar, em formato JSON, as *issues* dos repositórios da lista no banco de dados NoSQL MongoDB. Informações como o identificador, título, descrição, comentários, data de criação, nome do repositório e outras informações relevantes das *issues* foram extraídas. Esta mineração foi realizada no período de 4 de Junho à 24 de Julho de 2019 em um computador com especificações: processador Core i7-7500U, 8GB RAM, SSD 240 GB.

3.2. Pré-processamento da base de dados

O pré-processamento dos dados foi realizado para reduzir o ruído dos dados irrelevantes para o estudo, e assim preservar somente os termos das mensagens realmente descritas pelo usuário. Deste modo, os efeitos da interferência de elementos externos foram diminuídos, e.g., um trecho de código ou *exceptions* gerados pelo compilador e apresentado pelo usuário, ou uma *URL* que aponta para uma outra página.

Tal qual a atividade de mineração, implementamos um *script* em Python, utilizando a biblioteca RE[5]. A limpeza dos dados consistiu na remoção dos seguintes elementos: trechos de códigos, *tags* e códigos HTML, tabelas, *URL*, referências de tabela, duplicação de comentários, *Warnings*, *Exceptions*, nome de classes, caminhos de pastas e/ou arquivos, caracteres especiais, números, espaços em branco sucessivos e *stopwords*.

[3]Código-fonte disponível em https://github.com/joselitojunior94/gfetcher
[4]https://pygithub.readthedocs.io
[5]https://docs.python.org/3/library/re.html

3.3. Construção da coleção de termos técnicos

Para construir a coleção de termos, utilizamos a contagem de termos presentes nas *issues*, considerando os seguintes campos: título, descrição e comentários. Nessa atividade, identificamos que muitos termos existentes não eram relevantes para o estudo. Mesmo após a fase de remoção de *stopwords*, alguns termos não tinham ligação com a área. Para solucionar este problema e analisar os termos relevantes nesta base foi necessário construir um filtro, que reunisse uma lista de termos específicos de computação, programação e outras áreas afins para direcionar o estudo dos termos e padrões utilizados frequentemente nos repositórios. Os termos técnicos foram pesquisados e selecionados a partir de dicionários [6] [7] [8] online da área de computação. A validação de termos foi realizada através da interpretação dos significados e relevância dos termos para o estudo.

O estudo reuniu um conjunto de 577 termos específicos do domínio de computação para serem estudados na base de *issues*. Por meio da lista de termos selecionada anteriormente, mais uma fase de construção de termos foi realizada utilizando um script implementado em Python, que faz uso da biblioteca NLTK com o módulo *Wordnet* [9]. Um outro conjunto de termos foi encontrado. Entretanto, muitos não estavam alinhados com a palavra pesquisada, ou com o escopo do estudo. Então, foi realizada uma limpeza manual da tabela retornada pelo *script*. Ao todo 63 termos foram considerados e anexados a lista dos termos técnicos anteriormente estabelecida.

3.4. Processamento das frequências dos termos na base de dados

Uma *issue* é composta por alguns elementos que são importante no processo de tratamento. Assim, ela contém título, descrição e comentários realizados por usuários da plataforma. Neste estudo a análise dos termos foi divida em: estudo da frequência dos termos específicos de cada um desses termos.

A etapa de contagem da frequência foi realizada utilizando a biblioteca do NLTK que possui um módulo para estudo da frequência de palavras em *corpus* de texto. Uma estrutura com as palavras mais recorrentes e a sua frequência é retornada pela função. Esta estrutura representa todas as palavras com as ocorrências da base em questão. Assim, inicia-se o processo de consulta da lista dos termos específicos da área nesta estrutura de dados da frequência de termos da base. Com isso, é possível consultar todos os termos técnicos presentes e suas ocorrências. Um filtro para separar os termos mais frequentes dos títulos, descrições e comentários foi utilizado, considerando três estruturas de dados responsáveis por cada elemento da *issue*.

Após o retorno, os dados fornecem insumos para construção das tabelas e gráficos para as funções do script, utilizando as bibliotecas de tabelas CSV[10] e de gráficos Matplotlib[11], para posterior análise e discussão dos dados.

[6] http://foldoc.org/contents.html
[7] Definições de termos de *Tech Terms Computer Dictionary* https://techterms.com/
[8] https://www.webopedia.com/
[9] https://www.nltk.org/howto/wordnet.html
[10] https://docs.python.org/3/library/csv.html
[11] https://matplotlib.org/

4. Resultados encontrados

O estudo analisou 510.212 *issues* que, combinadas com os textos dos títulos, decrição e comentários apresentaram um *corpus* com 3.075.304 elementos textuais, com 60.991.362 termos utilizados por usuários e contribuidores de 85 repositórios do Github. Em uma lista de 608 termos específicos de computação, observamos como e onde esses termos se apresentam nas *issues*. A análise deste *dataset* gerou dados importantes para responder as perguntas de pesquisa, que serão discutidas a seguir. Todas as tabelas dos resultados e algoritmos utilizados podem ser encontrados em: **https://github.com/joselitojunior94/CTIC2020**.

QP1. Com o tratamento adequado dos dados dos 85 repositórios da base de dados e o estudo da frequência destes termos, os resultados apresentaram a ocorrência de termos técnicos na discussão dessas *issues*. Foram encontrados 566 termos da lista com pelo menos uma ocorrência. Apenas 42 termos não foram encontrados na base de dados. A Figura 2 mostra a ocorrência dos 25 termos.

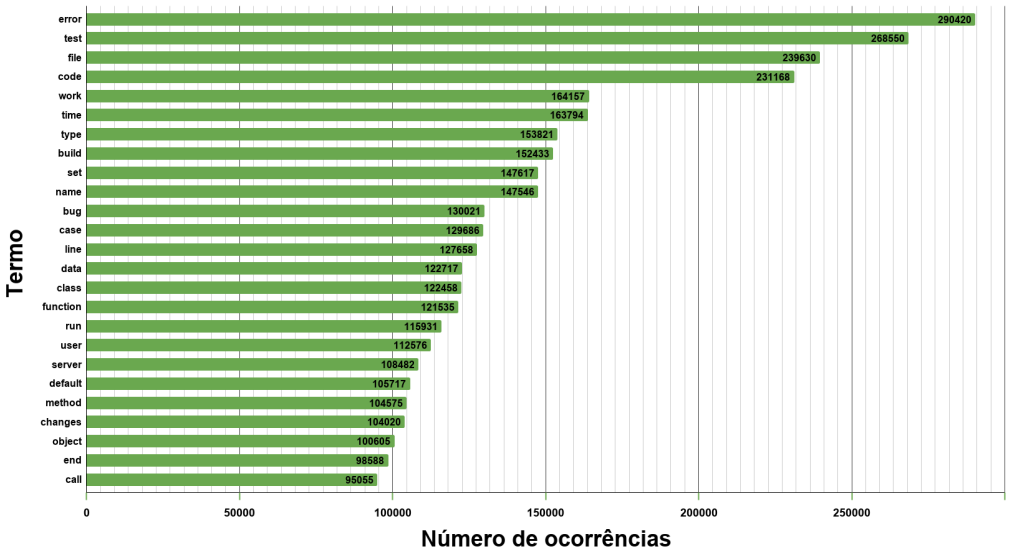

Figura 2. Frequência das palavras nas *issues*

QP2. Os títulos, descrições e comentários da *issue* foram analisados separadamente para investigar como a ocorrência dos termos se comportam nos diferentes campos.

Termos frequentes dos títulos: Seguindo a tendência dos resultados gerais, nos títulos das *issues* foram encontrados os termos mais frequentes, conforme apresentado na Figura 3. Os termos mais frequentes foram: *error* (16.707 ocorrências), *icon* (16.135 ocorrências), support (14.274 ocorrências) e *test* (12.192 ocorrências). Resultado semelhante no estudo geral das *issues* com todos os elementos reunidos, que apresentou *error* em primeiro e *test* também sendo apresentado entre os 4 primeiros termos mais citados. A palavra *documentation* também aparece entre os 25 termos mais citados, o que evidencia uma discussão não somente em código mas sobre a documentação nos repositórios.

Termos frequentes na descrição fornecida pelos usuários: Com a descrição dos defeitos fornecida pelos usuários da aplicação temos um comportamento semelhante ao título com a frequência da palavra *error*, com o maior número de ocorrências (143.151), seguido de *file* (113.064 ocorrências), e *test* (90.795 ocorrências). É válido ressaltar que há uma

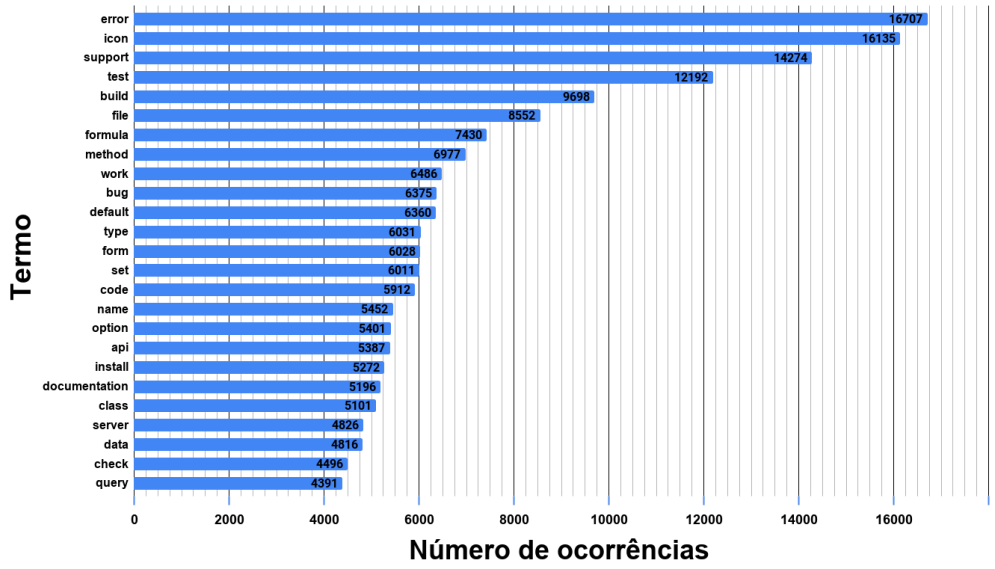

Figura 3. Frequência das palavras nos títulos da *issue*.

quantidade de termos técnicos como elementos de entidade de código em programação como: *type*, *function*, *data*, *class*, *method*, dentre outros. A Figura 4 apresenta os 25 termos mais recorrentes.

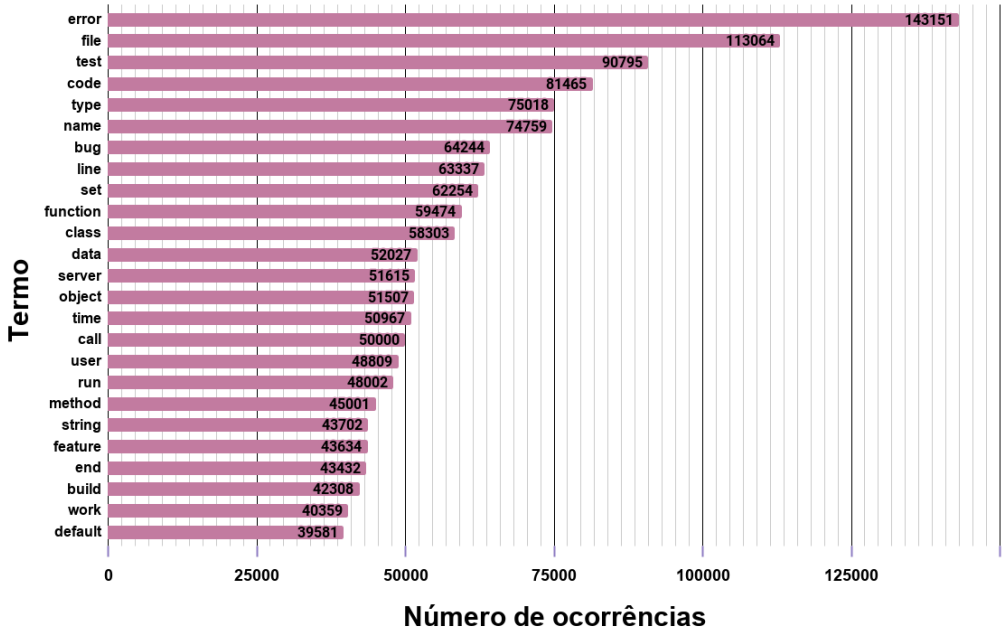

Figura 4. Frequência das palavras nas descrições dos defeitos fornecidos pelos usuários

Termos frequentes nos comentários respondidos aos usuários: Os resultados apresentados nos comentários refletiram outras frequências de termos. Nos comentários os contribuidores do projeto citaram com mais frequência os termos *test* (165.563 ocorrências), *code* (143.791 ocorrências) e *error* (130.562 ocorrências). O termo *error* nos outros elementos das *issues* foi mais recorrente do que nos comentários. A Figura 5 apresenta a

frequência de termos. Uma possível explicação é que os comentários podem refletir um vocabulário distinto, por ser uma resposta ou um feedback dos desenvolvedores para o devido erro. Assim, termos técnicos podem aparecer mais nas respostas do que no próprio texto de relato do defeito.

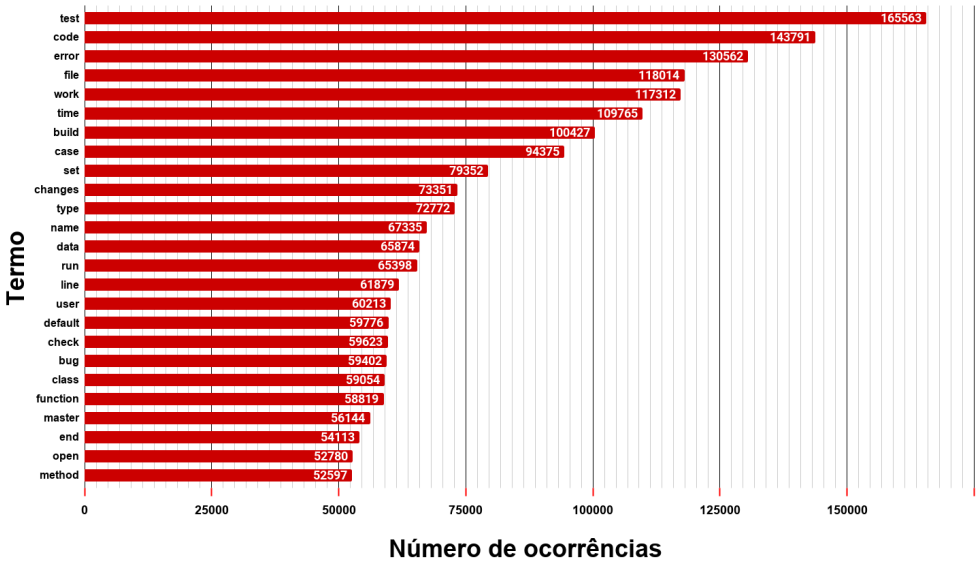

Figura 5. Frequência das palavras nos comentários dos contribuidores

O estudo da ocorrência desses termos pode trazer quais termos estão associados aos defeitos reportados nesses projetos. Para um desenvolvedor de software, compreender que o termo *Class* é citado no repositório pode ser um indicativo de alerta, no sentido de que é necessário atenção e cautela no tratamento de defeitos relacionados às classes do projeto. A ocorrência do termo *Data* também pode sugerir problemas relacionados à manipulação de dados ou banco de dados. Esses indícios podem iniciar um processo maior para conhecimento do repositório e das suas reais limitações ao resolver os defeitos, e possibilita à equipe de desenvolvimento implementar melhores estratégias para tratamento de defeitos reportados.

5. Ameaças à validade

Validade Interna: A lista do desafio *MSR Challenge Dataset* foi apresentada para a edição de 2014 da referida conferência. Alguns repositórios podem estar desatualizados. Por isso, um script de mineração foi construído para obter a base de dados mais atualizada possível. Repositórios ativos ainda possuem cadastro de novas *issues*. A base de dados reflete um recorte dos repositórios no período de mineração, sendo desconsideradas novas *issues* após o término da fase de obtenção dos dados. A construção da lista de termos específicos pode não conter todos os termos da área de computação. Para este propósito, mais de uma pesquisa, em bases diferentes, foi realizada.

Validade Externa: Os repositórios utilizados neste estudo podem apresentar características distintas. A generalização dos resultados para outra base de dados, sejam elas de repositórios diferentes do utilizado neste estudo, rastreador de defeitos diferente do Github e outros fatores que diferem, podem não apresentar o mesmo resultado encontrado utilizando esta configuração. Assim, esta base de dados foi escolhida por ter sido

utilizada em uma mais importantes conferências da área de Engenharia de Software (que discute aspectos de mineração de repositórios) e conter repositórios e quantidade de dados suficientes para condução de um estudo em larga escala.

Validade de Construto: A área de domínio dos repositórios pertencentes à base de dados é um fator que contribui para condução do resultado. Pode haver mais repositórios de um certo domínio e características semelhantes. Por isso que ao selecionar uma lista que apresenta repositórios de domínio distintos é realizada uma tentativa de aleatorização da base de dados para trazer uma maior fidelidade aos resultados. Entretanto, o estudo não contemplou discussões específicas por domínio de aplicação, restando tal assunto para trabalhos futuros.

6. Trabalhos Relacionados

Os erros de variabilidade do núcleo do Linux foram o foco do trabalho de [Abal et al. 2014]. Além disso foi realizado um estudo qualitativo de 42 bugs de variabilidade apresentados no repositórios de defeitos do Linux. Ao final, os autores apresentam uma lista categorizada com os defeitos apresentados e suas ocorrências.

O estudo de [Gromova et al. 2019] propôs uma uma abordagem utilizando vários indicadores para estudar e avaliar a qualidade da descrição dos defeitos utilizando uma ferramenta criada para o estudo. Foram extraídas 5.242 *bugs* reportados dos projetos do JBOSS no JIRA. A análise foi realizada utilizando o estudo dos defeitos reportados por módulos, prioridade e resolução dos defeitos através da ferramenta que utiliza aprendizado de máquina para análise dos textos.

O estudo de [Catolino et al. 2019] procurou entender, caracterizar e classificar defeitos reportados em *issues* dos projetos Mozilla, Apache e Eclipse. No estudo foram analisados 1.280 *bugs* que possibilitou a elaboração de uma taxonomia de tipos de *bugs* reportados. Após a construção, um modelo de classificação automática foi construído para encontrar os *bugs* mais comuns nestes repositórios.

Nestes trabalhos, o foco da análise foi em repositórios específicos, ou pertencentes à alguma família de *software*. Nosso trabalho reúne um dicionário de termos técnicos definidos nos dicionários de desenvolvimento e a ocorrência destes termos nas *issues* é estudada para encontrar os padrões de defeitos pelos termos técnicos da área.

7. Considerações Finais

Observar o comportamento das *issues* em uma plataforma de gerenciamento de projetos permite monitorar e antecipar-se a possíveis defeitos que a solução pode apresentar. Neste estudo foram analisadas 510.212 *issues* com 3.075.304 textos dos títulos, descrições e comentários que, juntos, compõem 60.991.362 palavras de 85 repositórios de projetos do Github. Com o auxílio de uma lista de 608 termos técnicos selecionados manualmente a partir de dicionários online, a frequência na utilização dos termos pelos contribuidores dos repositórios foi analisada.

Os resultados das ocorrências dos termos dos repositórios estudados apontam uma maior atenção na implementação de código com maior aplicação de termos relacionados a codificação, tais como: *file*, *code*, *type*, *class* e *outros*. O termo *test* foi o segundo termo mais relevante de ocorrências, trazendo presença de termos sobre testes nas *issues* nos

repositórios. Porém, a utilização de termos como *documentation* (Documentação) e *data* (Dados) apresentam outros tipos de discussões também em outras áreas. Os resultados deste estudo mostram que ao compreender a ocorrência dos termos nas discussões levam a indicativos de defeitos em áreas estruturais do projeto, como codificação, testes, tratamento dos dados e documentação. Tais indicativos são fundamentais para início da compreensão do ambiente de falhas, orientando os desenvolvedores a encontrar os indicativos de defeitos nas estuturas fundamentais do projeto, alocando assim recursos para melhor gerenciamento e manutenção.

Para trabalhos futuros, uma estratégia possível é a aplicação de métodos de aprendizado de máquina para detecção e interpretação do contexto dos termos. Uma outra direção futura é analisar os termos técnicos com análise de sentimentos, para investigar relações no emprego dos termos e o contexto da mensagem nas *issues*.

Agradecimentos: O presente trabalho foi realizado com apoio do CNPq (123368/2017-0, 116225/2018-1) e da FAPESB (JCB0060/2016), sendo continuado com apoio da Capes - Nº do Processo: 88887.494000/2020-00.

Referências

Abal, I., Brabrand, C., and Wasowski, A. (2014). 42 variability bugs in the linux kernel: a qualitative analysis. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE)*, pages 421–432.

Boechat, G., Júnior, J. M., Machado, I., and Mendonça, M. (2019). Análise de sentimentos em discussões de issues reabertas do github. In *Anais do VII Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pages 13–20. SBC.

Card, D. N. (1998). Learning from our mistakes with defect causal analysis. *IEEE software*, 15(1):56–63.

Catolino, G., Palomba, F., Zaidman, A., and Ferrucci, F. (2019). Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software*, 152:165–181.

Grady, R. B. (1996). Software failure analysis for high-return process improvement decisions. *Hewlett Packard Journal*, 47:15–24.

Gromova, A., Itkin, I., Pavlov, S., and Korovayev, A. (2019). Raising the quality of bug reports by predicting software defect indicators. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 198–204. IEEE.

Karahroudy, A. A. and Tabrizi, M. (2011). Software defect taxonomy, analysis and overview. In *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*.

Sullivan, M. and Chillarege, R. (1991). Software Defects and their Impact on System Availability: A Study of Field Failures in Operating Systems. In *International Symposium on Fault-Tolerant Computing*, volume 21, pages 2–9.

Zubrow, D. (2009). IEEE Standard Classification for Software Anomalies. *IEEE Computer Society*.