

Capítulo 6

Um Estudo sobre a Implementação de uma Extensão para FGRM

6.1 Introdução

Durante esta dissertação discutimos que as funcionalidades oferecidas FGRMs atendem as expectativas dos seus usuários. Todavia, após resultados como aqueles obtidos nos Capítulo 4 e 5, verificamos que existe um espaço para melhorias das funcionalidades existentes ou mesmo para a proposição de novas. Alguns estudos vêm seguindo esta tendência, especialmente explorando a capacidade de extensão propiciada por algumas FGRMs. A extensão *Buglocalizer* [Thung et al., 2014c], criada para a ferramenta Bugzilla, possibilita a localização dos arquivos do código fonte que estão relacionados ao defeito relatado. A ferramenta extrai texto dos campos de sumário e descrição da RM. Este texto é comparado com o código fonte por meio de técnicas de Recuperação da Informação.

Na mesma linha, o *NextBug* [Rocha et al., 2015] é uma extensão para o Bugzilla que recomenda novas RMs para o desenvolvedor com base na que ele esteja tratando atualmente. Na ferramenta proposta por Thung e outros [Thung et al., 2014b] o foco é na determinação de defeitos duplicados. A contribuição deste trabalho é a integração do estado da arte de técnicas não supervisionadas para detecção de RMs duplicadas.

Esta dissertação se propôs em contribuir com a melhoria das funcionalidades das FGRMs mediante a apresentação e discussão de um conjunto de recomendações no Capítulo 5. Apesar de ter sido conduzido um processo de avaliação das recomendações, cujo resultado demonstrou uma boa aceitação dos participantes, optamos por analisar o impacto da implementação de uma das sugestões propostas.

Idealmente gostaríamos de transformar todas as sugestões em extensões de funcionalidades para as FGRMs. Não há razões que justifiquem a priorização de implementação de uma recomendação sobre outra. Entretanto, após alguns ensaios e combinando de maneira mais intuitiva do que seguindo um fluxo de critérios, foi investido mais esforço no desenvolvimento de uma extensão para o suporte à qualidade de relato. A extensão proposta tem por objetivo ser uma Prova de Conceito, ou seja, demonstrar com o fim de verificar que conceito proposto possui certo *potencial* prático. Na próxima seção apresentamos uma breve discussão sobre o problema da baixa qualidade do relato no contexto das FGRMs.

6.2 Qualidade do Relato de uma RM

É possível considerar que as informações mais relevantes estejam no relato de uma RM, que é o atributo que representa o texto redigido pelo Reportador. Sabemos que o ato de reportar uma RM pode ser realizado por um usuário do software ou por um membro da equipe de desenvolvimento ou manutenção. Por esta razão, podemos encontrar Reportadores com diferentes níveis de conhecimento sobre o sistema. Esta situação pode provocar um efeito colateral: a baixa qualidade do texto no relatado de uma RM, como por exemplo a falta da informação necessária para sua solução.

Alguns estudos afirmam que a baixa qualidade do relato prejudicam o andamento do projeto mais do que RMs duplicadas [Bettenburg et al., 2007]. No estudo realizado por Bettenburg e outros [Bettenburg et al., 2008b] foi desenvolvido um levantamento com questionário (*survey*) entre desenvolvedores e usuários de três projetos de código aberto¹. O objetivo era coletar informações de modo a verificar o que produziria um bom relato. Os resultados demonstraram que, do ponto de vista dos desenvolvedores, são consideradas informações úteis para estar no relato de uma RM:

- a sequência de ações executadas até o aparecimento do erro (se for o caso), também conhecida como *etapas para reproduzir*;
- o registro de pilhas de ativação (stack traces) que são arquivos com os históricos de chamada de métodos (logs) que ocorreram antes do aparecimento do erro.

No estudo proposto por Zimmermann e outros [Zimmermann et al., 2009b] é discutida a importância de que a informação descrita em uma RM seja relevante e completa a fim de que o problema relatado seja resolvido rapidamente. Contudo, na prática,

¹Apache (<http://www.apache.org/>), Eclipse (<https://www.eclipse.org>) e Mozilla (<https://www.mozilla.org>)

a informação apenas chega ao desenvolvedor com a qualidade requerida após diversas interações com o usuário afetado. Com o objetivo de minimizar este problema os autores propõe um conjunto de diretrizes para a construção de uma extensão capaz de reunir informações relevantes a partir do usuário além de identificar arquivos que precisam ser corrigidos para resolver o defeito.

Conforme exposto, a utilização de uma extensão que suporte a melhoria da qualidade do relato pode resultar nas seguintes melhorias: redução no tempo necessário para análise do que foi solicitado na RM; facilidade na identificação de RMs duplicadas; disciplinar os Reportadores sobre a boa prática de fornecer um relato com maior qualidade. Estas vantagens têm impacto no custo e qualidade do software produzido.

6.3 Uma Extensão para Suporte à Qualidade do Relato

Considerando as vantagens de analisar a qualidade do relato de uma RM, desenvolvemos uma extensão para o módulo de gerenciamento de RMs da plataforma Github². De uma maneira geral, a extensão proposta realiza a análise do relato contido em uma RM e com base nele gera um comentário com “dicas” que podem melhorar a qualidade das informações prestadas. Nas próximas seções apresentamos com mais detalhe o desenho e funcionamento da extensão proposta.

6.3.1 Desenho da Extensão

O objetivo da extensão é analisar de maneira automatizada a qualidade do relato de uma *issue* em repositórios do GitHub. No contexto da extensão proposta, o elemento correspondente ao conceito de uma RM foi mapeado para o elemento *issue* no âmbito da plataforma Github. Um Repositório é o elemento mais básico do GitHub e contém os arquivos do projeto (incluindo documentação) e armazena o histórico de revisões de cada arquivo³. A execução da extensão resulta em um conjunto de dicas para o responsável por redigir a *issue* com o intuito de melhorar a qualidade da informação fornecida no relato, por esta razão recebeu o nome de *IssueQuality*.

²<https://guides.github.com/features/issues/>

³<https://help.github.com/articles/github-glossary/>

6.3.1.1 Visão Geral

A extensão proposta pode ser vista como um cliente para API do Github⁴ que possibilita analisar a qualidade da informação fornecida no relato. Uma visão geral sobre o funcionamento da *IssueQuality* pode ser visualizada na Figura 6.1. A partir de uma lista pré-definida de repositórios (1) a extensão solicita, através da API do Github (2), o conjunto de *issues* que estão com a situação “aberta” (etapas 3 e 4). Para cada uma das *issues* recebidas, a ferramenta cria um comentário por meio da API (5) que é registrado e armazenado na base de dados do Github (etapas 6 e 7). A partir do comentário gerado o próprio Github se encarrega de notificar (8) o responsável por relatar a *issue* (9).

A partir desta notificação espera-se que o responsável inclua a informação solicitada mediante a criação de um novo comentário.

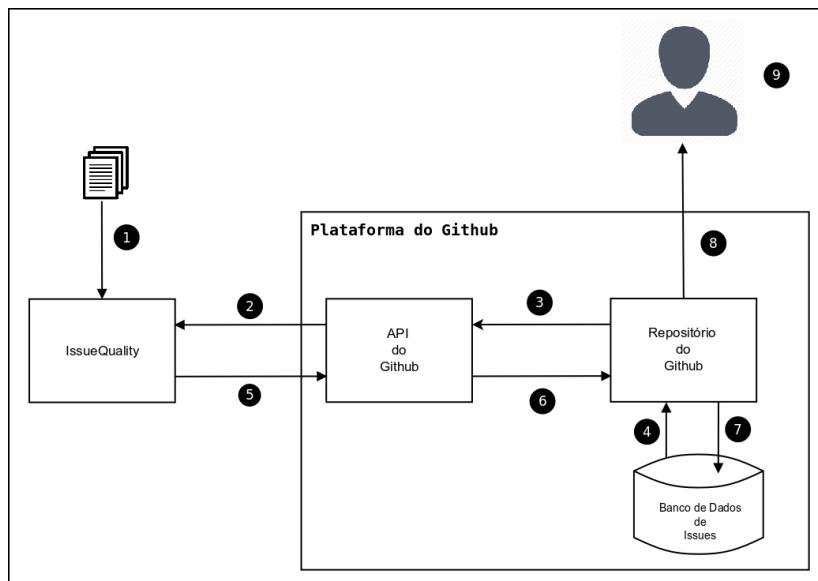


Figura 6.1: Visão geral do funcionamento da extensão *IssueQuality*

Para gerar o comentário descrito anteriormente, a extensão avalia alguns atributos do texto que compõe o relato da RM. Os detalhes de como estes atributos são analisados e o comentário construído estão descritos na próxima seção.

6.3.1.2 Análise da Qualidade do Relato

Para cada issue analisada a extensão cria um *vetor de características* que armazena uma pontuação para cada atributo do texto que será analisado. Estes valores podem ser binário (por exemplo, anexo presente ou não) ou contínuo (por exemplo, legibilidade

⁴<https://api.github.com/>

do texto). A análise dos atributos utilizam da sintaxe da linguagem de marcação Markdown⁵, que é o padrão para as issues dos repositórios no GitHub.

Conforme descrito, o resultado da análise feita pela extensão é um comentário na *issue*. Em geral, ele é composto de três partes: *cabeçalho*, *corpo* e *dicas*. O cabeçalho apresenta um texto padrão que é personalizado com o nome do usuário (*login*) no Github do reportador. Ao utilizarmos a sintaxe *[Github login]* o próprio Github se encarrega de enviar um e-mail notificando o usuário sobre o comentário. A Figura 6.2 exibe o cabeçalho padrão incluídos nos comentários.

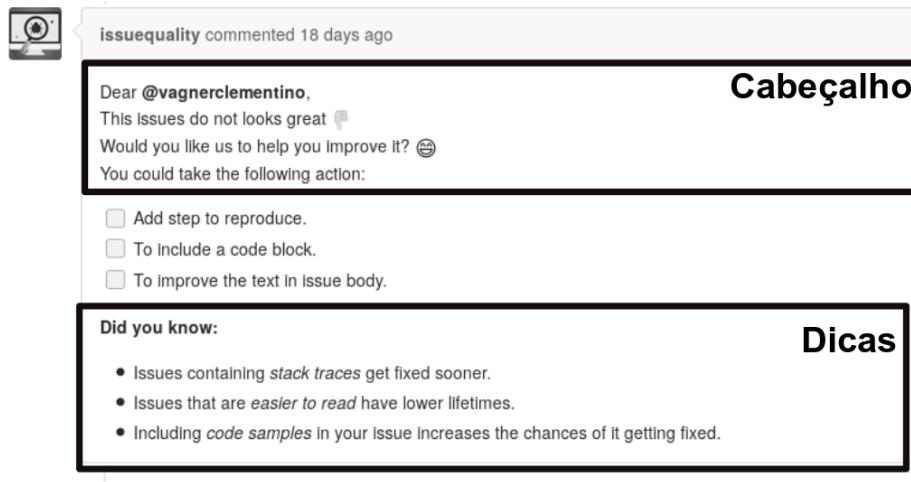


Figura 6.2: Comentário produzido pela extensão IssueQuality com os cabeçalhos e dicas padrões.

Ao final do comentário é incluído um conjunto de dicas com objetivo de reforçar com reportador os benefícios que a melhoria da qualidade do relato pode ter na solução de sua *issue*, como por exemplo dizendo que issues que são mais fáceis de serem lidas possuem um tempo de solução menor. Estas dicas foram obtidas com base na literatura sobre melhoria da qualidade do relato, especialmente nos trabalhos de Bettenburg e outros [Bettenburg et al., 2007, Bettenburg et al., 2008b]. Na Figura 6.2 é possível visualizar como algumas dicas são apresentadas.

O corpo é a parte dinâmica do comentário. Ele é construído incluindo fragmentos de texto quando certos critérios de aceitação não foram atendidos. Por exemplo, caso não seja detectada a presença de “*etapas para reproduzir*” no relato de uma *issue* o seguinte fragmento de texto é incluído no corpo do comentário: “*Add step to reproduce*”. Os atributos avaliados e os critérios de aceitação estão descritos na Tabela 6.1.

O corpo do comentário é produzido com análise dos seguintes atributos do relato da RM: *Etapas para Reproduzir*, *Arquivos Anexados*, *Fragmentos de Código*, *Com-*

⁵<https://help.github.com/categories/writing-on-github/>

Atributo	Critério de Aceitação
Completude de Palavras-chaves	Existência de uma lista representando as etapas executadas até a ocorrência do erro.
Arquivos Anexados	Pelo menos um arquivo anexado à issue
Fragments de Código	Existência de pelo menos um fragmento de código no relato da issue.
Completude do Texto	As palavras que compõem o relato da issue devem fazer parte de pelo menos duas categorias.
Legibilidade do Texto	Dois testes de legibilidade apresentarem valores acima dos limiares.

Tabela 6.1: Critérios de aceitação e forma de análise utilizados na análise de qualidade do relato.

pletude do Texto e *Legibilidade do Texto*. Estes atributos foram baseados no estudo realizado por Bettenburg e outros [Bettenburg et al., 2008b]. A seguir apresentamos os detalhes de como cada atributo é avaliado.

Etapas para Reproduzir: Verifica se o reportador incluiu uma lista, na forma de itens, descrevendo as etapas executadas até a ocorrência da falha. Para detectar este padrão a extensão aproveita da linguagem Markdown, que é o padrão utilizado para redigir o relato da *issue*, e que possui uma sintaxe pré-definida para listas. O padrão é detectado através da utilização de expressões regulares. Pode ocorrer que o Reportador utilize outro formato para relatar as etapas executadas até a ocorrência da falha, contudo, o fato da extensão exigir a informação através de uma lista, pode criar no Reportador uma boa prática. Cabe ressaltar que uma lista com etapas para reproduzir foi uma das informações mais relevantes a estarem no relato de uma RM [Bettenburg et al., 2008b].

Arquivos Anexados: Nesta dimensão avaliamos a existência de arquivos anexados à *issue*, tais como capturas de telas (screenshots) e cadeia de registros de ativação de funções (stack trace). A detecção é realizada utilizando expressões regulares que é a sintaxe padrão para o relato da RM⁶. Não houve avaliação sobre o conteúdo do anexo, contudo, conforme descrito na Tabela 6.1, uma mensagem é incluída no corpo do comentário no caso de nenhum anexo ser detectado. A existência de anexos em uma RM consta como uma das informações mais relevantes do ponto de vista dos desenvolvedores [Bettenburg et al., 2008b].

Fragments de Código Este atributo de avaliação verifica se fragmentos de código foi adicionado no relato da *issue*. O processo de detecção faz uso de expressão regulares e da sintaxe oferecida pela versão do Markdown utilizado pelo Github⁷. De forma

⁶<https://guides.github.com/features/mastering-markdown/>

⁷<https://guides.github.com/features/mastering-markdown/#GitHub-flavored-markdown>

similar aos atributos descritos anteriormente, a verificação de fragmentos de código fonte é binária, ou seja, foi avaliada a existência ou não de trecho de código.

Completure do Texto: Nesta dimensão da avaliação há uma premissa que possa existir um vocabulário comum no relato de diferentes RMs. Sendo assim, algumas palavras aparecem com determinada frequência no relato de uma RM, independente do projeto. Com o objetivo de compreender como as pessoas descrevem problemas de software, Ko e outros [Ko et al., 2006] analisaram o título de 200.000 RMs de cinco projetos de código aberto desenvolvidos na linguagem Java. Nós reutilizamos esta base de dados⁸ para construir uma distribuição da frequência de ocorrências de palavras no relato de uma RM. Em uma primeira etapa, removemos as palavras de parada (stop-words), reduzimos as vocábulos⁹ e selecionamos as 100 palavras com maior frequência. Em seguida, categorizamos as palavras nos seguintes grupos:

- itens de ação (do, work, open)
- relacionado com compilação (build, task)
- relacionado com documentação (support, help, content)
- comportamento esperado ou observável (fail, error, crash)
- relacionado com projeto (management, list)
- relacionado com código fonte source code-related (java, code, method)
- elementos da interface do usuário (menu, display, button)

Legibilidade do Texto Por fim a extensão avalia o nível de legibilidade do texto com base em testes largamente utilizados na literatura [Si & Callan, 2001]. Os testes são formulados para avaliar a legibilidade do texto, geralmente contando sílabas, palavras e frases. Neste estudo utilizamos os testes de legibilidade *Flesch–Kincaid*, *Automated Readability Index - ARI* e *Dale–Chall Readability Formula*. As avaliações foram selecionadas por apresentarem metodologias distintas para determinar a legibilidade do texto.

O Flesch-Kincaid (FK) é baseado no número de sílabas das palavras que compõem as sentenças do texto. Existem dois testes, o Flesch Reading Ease e Flesch–Kincaid

⁸Disponível para download em <http://www.cs.cmu.edu/~marmalade/reports.html>.

⁹Do inglês *stemming* é o processo de reduzir palavras flexionadas (ou às vezes derivadas) ao seu tronco (stem), base ou raiz, geralmente uma forma da palavra escrita. Por exemplo um algoritmo de stemming reduz as palavras “fishing”, “fished” e “fisher” para a raiz “fish”.

Grade Level, que diferem pelo fator de ponderação utilizado. A extensão utilizou o Flesch Reading Ease que recebe um texto em língua inglesa e retorna um valor inteiro representando a dificuldade de entendimento. Pontuações mais altas indicam que o material é mais fácil de ler [Kincaid et al., 1975]. Desta forma, foi considerada como legibilidade baixa uma pontuação *menor do 50*. Este valor foi baseado em uma tabela pré-definida pelo próprio teste [Kincaid et al., 1975].

O ARI considera o número de caracteres de cada palavra. O resultado do teste é uma representação aproximada do grau de escolaridade americano necessário para compreender o texto [Senter & Smith, 1967]. De maneira aproximada, o grau 1 corresponde a idades 6-8. O nível de leitura 8 corresponde a jovem de 14 anos. O grau 12, o mais alto no ensino secundário dos EUA, corresponde ao nível de leitura de 17 anos de idade.

Por outro lado, o teste Dale-Chall é baseado em um conjunto mínimo de palavras. A fórmula usa uma lista de 3000 vocábulos que grupos de estudantes americanos de quarta série poderiam entender de forma confiável, partindo-se da premissa que qualquer palavra nessa lista não seja de difícil compreensão [Dale & Chall, 1948]. No caso dos testes ARI e Dale-Chall a legibilidade será considerada ruim se o valor do teste *for maior ou igual a 13*, ou seja, uma pessoa deveria estudar no mínimo 13 anos para “entender”. Esta limiar foi utilizado no trabalho de Bettenburg e outros [Bettenburg et al., 2008b].

6.4 Execução da Extensão em Projetos Reais

A extensão descrita neste Capítulo foi desenvolvida como prova de conceito. Neste sentido, o desenho e desenvolvimento teve foco na viabilidade do que estava sendo proposto do que melhorar ou sobrepor determinada solução do estado da arte. Entretanto, com o objetivo de avaliar a extensão em um contexto mais próximo do real, ela foi executada utilizando os dados de *issues* de projetos código aberto hospedados no Github.

6.4.1 Seleção dos Projetos

Para realizarmos a execução da extensão optamos por uma amostra de conveniência composta de projetos hospedados na plataforma Github. Para isso utilizamos os mesmos critérios descritos na Seção 5.3.1. As exceções foi que incluímos a restrição de que o projeto fosse desenvolvido em Java e avaliamos os dez primeiros projetos ordenados pelo critério “most stars”. O condicionante da linguagem Java é por conta de algumas

decisões tomadas durante o desenvolvimento da extensão que exigem a utilização da linguagem. Após aplicação dos critérios descritos obtivemos os projetos apresentados na Tabela 6.2.

Projeto	Revisões	Ramificações	Lançamentos
elasticsearch	27.118	94	174
guava	4.081	84	175
spring-framework	14575	12	106

Tabela 6.2: Projetos utilizados no testes de execução da extensão. Os dados apresentados tem como referência 23/04/2017.

6.4.1.1 O Processo de Execução

Para avaliarmos a extensão começamos por criar uma ramificação (fork) dos projetos escolhidos. Este processo é realizado de forma automatizada pelo Github, contudo, não realiza a cópia das *issues*, que no caso deste teste é a informação mais relevante. Para transpor os dados das issues foi desenvolvido um processo automatizado que copiava o título e o relato (corpo da issue) do projeto original para a sua ramificação. Para cada projeto realizamos a cópia de 100 *issues*. A Figura 6.3 exibe uma *issue* do projeto Guava no lado esquerdo e sua cópia na ramificação criada. Como você pode observar o título e o relato são idênticos.

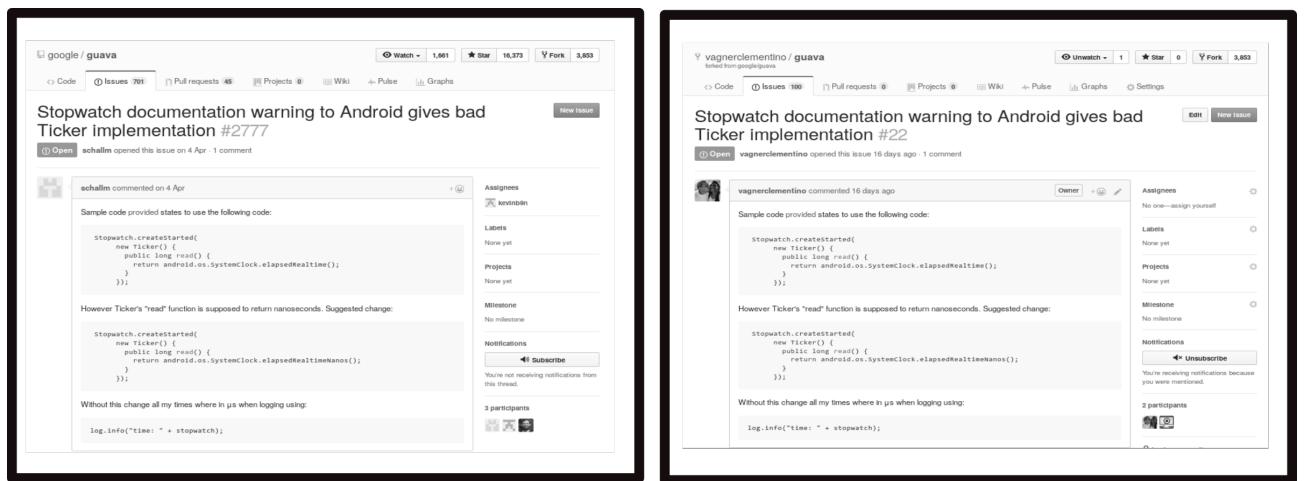


Figura 6.3: Cópia de uma issue do projeto Guava para a sua ramificação.

Após finalizado o processo de cópia executamos a extensão nas 100 *issues* de cada projeto. Na Figura 6.4 visualizamos o comentário gerado para a *issue* apresentada na Figura 6.3. Na próxima seção apresentamos algumas métricas do processo de

execução da extensão nos projetos escolhidos. Como se trata de uma prova de conceito estes dados podem nos ajudar na avaliação da viabilidade de implantação da extensão proposta.

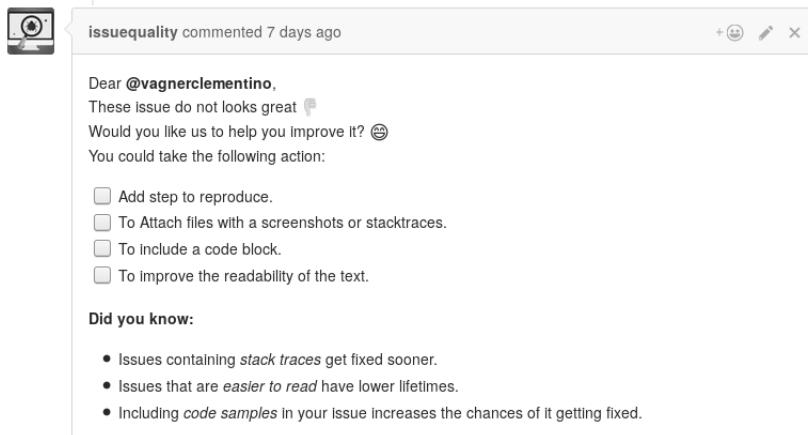


Figura 6.4: Comentário para a issue do projeto Guava exibida na Figura 6.3

6.4.2 Resultados

Uma possível preocupação com a inclusão deste tipo de extensão em uma FGRM é o atraso que possa ocorrer no processo de solução das RMs. É natural que um tempo adicional seja necessário para analisar a qualidade do relato. A Tabela 6.3 exibe o tempo em segundos que a extensão levou para analisar as *issues*. É possível verificar que em média cerca de 02 segundos são necessários para avaliar a qualidade do relato. Em alguns casos o processo é finalizado em menos de 01 segundo. Entretanto, é possível verificar situações em que o tempo de retorno da extensão é cerca de 30 segundos. Em alguns casos esta discrepância é maior, como no caso do projeto *Guava* que possui um desvio padrão acentuado onde o tempo de execução pode variar de 01 a 31 segundos.

Projeto	Média (seg.)	Mediana (seg.)	Desvio Padrão (seg.)	Min (seg.)	Max (seg.)
Todos	1,91	1	2,93	0	31
spring-framework	2,36	1	4,13	1	31
guava	1,71	1	1,9	0	16
elasticsearch	1,66	1	2,24	0	16

Tabela 6.3: Tempo de execução da extensão.

Com a execução utilizando os dados de *issues* de projetos reais gostaríamos de verificar em qual atributo do relato os problemas são mais frequentes. Conforme apresentado na Tabela 6.1 a extensão avalia os seguintes atributos do relato: Etapas

para Reproduzir, Arquivo Anexado, Fragmentos de Código, Completude de Palavras-Chaves e Legibilidade do Texto. Um comentário é gerado caso o relato não atenda a determinados critérios para cada atributo analisado. A Figura 6.5 exibe a frequência que um item foi incluído ao comentário para atributo.

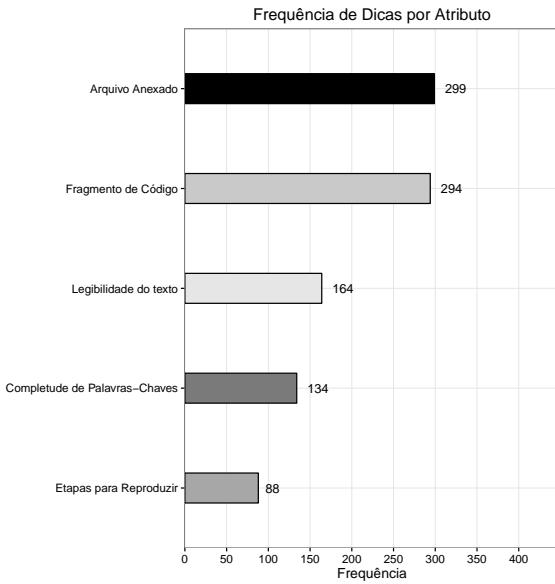


Figura 6.5: Frequência de dicas por atributo analisados.

Verificamos na figura que em quase todas as issues analisadas não foi detectado a inclusão de arquivos anexados ou de fragmentos de código. É possível observar que mais da metade do texto relatado apresentou uma legibilidade ruim do ponto de vista da nossa extensão. A situação menos frequente está relacionada com a inclusão de “Etapas para Reproduzir”, em que é avaliada a existência de uma lista detalhando os passos executados até a ocorrência da falha.

Projeto	Média (seg)	Mediana (seg)	Desvio Padrão (seg)	Min (seg)	Max (seg)
Todos	3,26	3	0,96	1	5
elasticsearch	2,95	3	0,85	1	5
guava	3,16	3	0,88	2	5
spring-framework	3,68	4	1,01	1	5

Tabela 6.4: Número de dicas retornadas pela extensão.

Durante a execução estávamos interessados em avaliar o número de dicas de melhorias na qualidade do relato a extensão gerou. Esta informação está exibida na Tabela 6.4. É possível observar que em média 03 dicas são incluídas em cada *issues*, ou seja, três em cada cinco dos itens apresentaram algum tipo de problema. Entretanto, para que possamos extrapolar qualquer discussão sobre este resultado era necessário

uma avaliação de pessoas envolvidas nos projetos utilizados de modo a detectar falsos positivos.

6.4.3 Discussão

As execuções que foram realizadas têm como foco avaliar a viabilidade da extensão proposta. Neste sentido, os resultados obtidos na Seção 6.4 devem ser avaliados sobre esta ótica. No caso do tempo necessário verificamos que em média não causa impacto no processo de solução de uma RM. Nos casos de maior duração, como por exemplo 31 segundos, pode estar relacionado a arquitetura escolhida. Conforme exposto, a extensão utiliza uma API do Github que pode estar sujeita a questões relativas à rede ou mesmo de sobrecarga.

Com relação aos problemas mais frequentes uma tendência dos usuários é em não incluir anexos, mas de relatar o problema copiando, por exemplo, os registros de pilha de ativação (stacks traces) no próprio corpo da *issue*. Por outro lado, um fator a destacar é que mais da metade das *issues* analisadas apresentam um texto com baixa legibilidade. Esta situação já foi verificada em estudos anteriores [Ko et al., 2006, Bettenburg et al., 2007]. Esta métrica depende menos de uma análise de alguém vinculado ao projeto pelo fato de utilizar índices existentes e utilizados em diversas áreas para medir legibilidade. Neste sentido pode ser interessante ao projetos de software disciplinar os Reportadores em fornecer sempre que possível um relato mais simples e direto.

6.5 Limitações e Ameças à Validade

O desenvolvimento da extensão possui limitações que ameaçam a sua validade. Inicialmente não é possível determinar se os atributos avaliados são os mais indicados para medir a qualidade do relato. Eles foram utilizados com base em estudos já realizados [Bettenburg et al., 2007] em que os atributos foram baseados na opinião de desenvolvedores.

Para que uma dica de melhoria seja disparada é necessário algum dos critérios descritos na Tabela 6.1 não seja atendido. No caso do atributo “Completeness of Palavras-Chaves” utilizou o conjunto de termos do trabalho de Ko e outros [Ko et al., 2006]. Apesar de ambos os estudos utilizarem projetos desenvolvidos na linguagem Java não podemos garantir que o conjunto de termos é o mesmo nas RMs de diferentes projetos.

A legibilidade do texto foi realizada utilizando testes descritos na literatura. Em todos os testes existem os limiares que definem o grau de legibilidade de um texto.

Entretanto, não sabemos qual o impacto de utilizarmos estes valores no relato de uma falha de software, por exemplo. Em geral, os testes de legibilidade são bastante genéricos, contudo, certas adequações podem ser necessárias para serem utilizados na análise do relato de uma RM.

A principal limitação deste estudo está na extração dos resultados. O ideal é que a execução da extensão fosse avaliada por um profissional vinculado aos projetos utilizados. Esta análise poderia ajudar na detecção de falsos positivos, por exemplo. Estendemos a importância deste tipo de suporte, mesmo que por inspeção de alguns casos. Um trabalho futuro desta dissertação seria reaplicar a extensão em uma configuração com suporte de um oráculo ligado ao projeto analisado.

6.6 Conclusão

Por se tratar de uma prova de conceito a extensão proposta apresentou um desempenho satisfatório. Em média, o tempo necessário para analisar a qualidade do relato não tem impacto no processo de solução das RMs. Além disso, foi possível analisar os diversos atributos o qual a extensão se propôs. Os resultados não podem ser extrapolados por não terem sido avaliados pelos desenvolvedores vinculados aos projetos. Todavia, apesar das limitações, uma extensão conforme a proposta pode disciplinar os Reportadores a fornecer relatos com mais qualidade.

