IssueQuality: Uma extensão para a análise da qualidade do relato de uma issue

O que foi feito

Visão Geral

O IssueQuality pode ser visualizado como um cliente para API do Github que possibilita analisar a qualidade da informação fornecida no relato de uma Requisição de Mudanças (RM) registrada em determinado repositório. O elemento correspondente ao conceito de RM na dissertação corresponde ao elemento *issue* no contexto da plataforma Github. 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 gera um comentário novamente utilizando a API (5) que é registrado e armazenado na base de dados do Github (etapas 6 e 7). A partir do comentário gerado a própria plataforma do Github se encarrega de notificar (8) o responsável por relatar a *issue* (9). A Figura 1 exibe uma visão geral do funcionamento da extensão proposta.

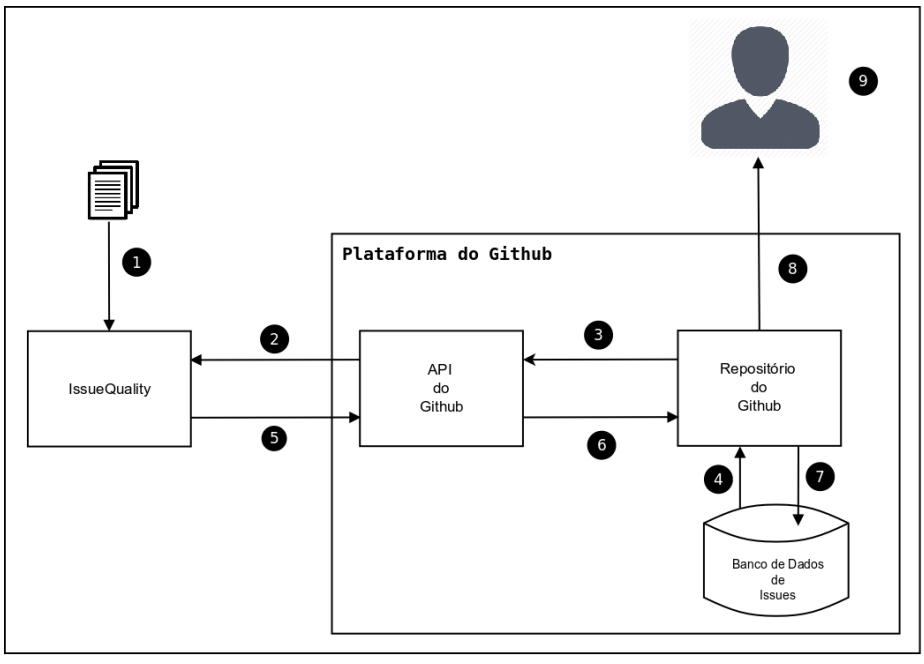

Figure 1: Visão geral do funcionamento da IssueQuality

Análise da Qualidade do Relato

Conforme exposto o comentário apresenta um conjunto de recomendações visando melhorar a qualidade do relato existente na *issue*. As sugestões são produzidas avaliando os seguintes atributos:

- *Etapas para Reproduzir:* Verifica se reportador incluiu uma lista, na forma de itens, descrevendo as etapas executadas até a ocorrência da falha.
- *Arquivos Anexados:* Avalia a existência de arquivos anexados à issue, como por exemplo screenshots ou stack trace.
- *Fragmentos de Código:* Verifica se algum fragmento de código foi adicionado ao relato da issue.
- *Completude de Palavras-Chaves:* Utilizando o conjunto de dados fornecido por Andy Ko e outros[1] foi construído a distribuição de frequências das palavras que ocorrem em uma issue. Em uma primeira etapa, removemos as palavras de parada (stopwords), reduzimos as palavras e selecionamos as 100 palavras com maior frequência. Em seguida, categorizamos as palavras escolhidas nos seguintes grupos:

  ```
  - action items  (do, work,open)
  - build-related (build, task)
  - documentation-related  (support, help, content)
  - expected and observed behavior (fail, error, crash)
  - project-related  (management, list)
  - source code-related (java, code, method)
  - user interface elements  (menu, display, button)
  ```

- *Legibilidade do Texto:* Avalia o nível legibilidade do texto com base em testes já existentes na literatura. Neste estudo utilizamos os testes de legibilidade Flesch–Kincaid, Automated Readability Index - ARI e Dale–Chall Readability Formula. Os testes foram selecionados por apresentarem metodologias distintas para determinar a legibilidade do texto. O Flesch-Kincaid é baseado no número de sílabas das palavras que compõem as sentenças do texto. O ARI considera o número de caracteres de cada palavra. Por outro lado, o teste Dale-Chall é baseado em um conjunto mínimo de palavras. Para o texto Flesch-Kincaid a legibilidade será considerada baixa para uma pontuação *menor do 50*. Para os outros testes, a legibilidade será considerada ruim se o número de anos de estudos necessário para o entendimento *for maior ou igual a 13*, conforme utilizado no trabalho de Bettenburg e outros[2].

[1] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In Proceedings of the 2006 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2006), pages 127–134, 2006

[2] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008, November). What makes a good bug report?. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 308-318). ACM.

Funcionamento

Para cada issue analisada a extensão cria um *vetor de características* que armazena uma pontuação para cada um dos itens listados anteriormente. Estes podem ser binários (por exemplo, anexo presente ou não) ou contínuo (por exemplo, legibilidade do texto). A análise dos atributos utilizam da sintaxe da linguagem de marcação Markdown, que é o padrão para as issues dos repositórios no GitHub.

O comentário produzido 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. Utilizando esta sintaxe o próprio Github se encarrega de enviar um e-mail notificando o usuário sobre o comentário.

Ao final do comentário é incluído um conjunto de dicas com objetivo de apresentar ao 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 de trabalhos acadêmicos sobre o assunto, especialmente o trabalho Bettenburg e outros[3]. A Figura 3 exibe o conteúdo das dicas que fazem parte do comentário.

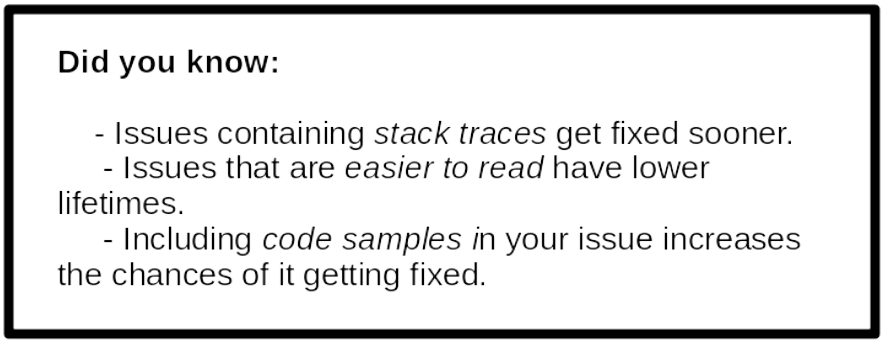

Figure 2: Dicas apresentadas ao reportador

O corpo é parte mais dinâmica do comentário. Ele é construído incluindo fragmentos de texto quando certos critérios de aceitação não foram atendidos. Por exemplo, caso não seja detectado 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*. O exemplo de utilização da extensão pode ser visualizado no repositório *vagnerclementino/flask* clique aqui.

[3]Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008, November). What makes a good bug report?. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering (pp. 308-318). ACM.

Capítulo da Dissertação

No capitulo de implementação a estrutura do texto será a seguinte:

```
- **Problema da Qualidade do Relato da RM**: breve relato do problema de
avaliar a qualidade do relato com base em alguns trabalhso sobre o assunto.
- **Desenho e Detalhes de Implementação**: apresenta o desenho e detalhes de
como a extensão foi implementada. Nesta parte também serão discutidos alguns
conceitos que foram utilizados.
- **Avaliaçaõ da extensão**: descreve o processo de avaliação da extensão
proposta. O plano é utilizar 04 projetos que utilizem a linguagem Java que
tenham em sua base pelo menos 100 *issues* em sua base. Após a execução da
extensão nas *issues* dos projetos seriam produzidos screenshots dos
comentários gerados. Estas imagens seriam apresentados aos colaboradores dos
projetos de modo que eles avaliem cada comentário gerado.
```