Análise estática de problemas no código-fonte de projetos públicos

Vinícius Luiz do Amaral[1]

Programa de Pós-Graduação em Modelagem Matemática e Computacional
Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG
viniciusluiz.doamaral@gmail.com[1]

Abstract. O objetivo deste trabalho é a análise da qualidade do código-fonte de projetos públicos com resultados disponíveis na SonarCloud. Neste estudo são considerados os seguintes tipos de problemas: *Bugs*, *Code Smells* e Vulnerabilidades. Os dados foram extraídos da API pública disponibilizada pela ferramenta dos quais, após a realização de um tratamento, foi possível obter alguns indicadores. Os resultados são apresentados em termos da proporção da ocorrência de cada problema identificado pela ferramenta por nível de gravidade bem como o detalhamento dos problemas mais recorrentes. Entende-se que a aplicação da metodologia proposta nos projetos de uma organização real pode auxiliar na identificação de oportunidades de melhorias por parte do time de desenvolvimento para que sejam combatidos os principais problemas reportados pela ferramenta.

Keywords: Qualidade de *Software*, SonarQube, SonarCloud, *Bugs*, *Code Smells*, Vulnerabilidades

1 Introdução

O conceito de qualidade de *software* é abrangente e deve ser observado não somente em características perceptíveis aos usuários, como o nível de acessibilidade ou o desempenho proporcionados durante o uso do sistema. Aspectos relacionados à estrutura interna tais como a organização e complexidade do código, vulnerablidade a problemas de segurança e a ocorrência de erros em fluxos de execução primários ou alternativos também devem ser considerados pois tendem a impactar na utilização e/ou evolução daquilo que fora construído [1].

O SonarQube é uma ferramenta para revisão de código e tem por objetivo realizar a detecção automática de Bugs[1], *Code Smells*[2] e Vulnerabilidades[3]. Para isto, esta ferramenta é capaz de avaliar se o código desenvolvido viola um conjunto de regras que servem como indicadores de qualidade. Atualmente existem

[1] Ocorrência de um erro no código que pode levar a um comportamento inesperado em tempo de execução.
[2] Problema de manutenibilidade que torna o código confuso e difícil de ser mantido.
[3] Problema de segurança que torna o código aberto à realização de ataques maliciosos.

regras implementadas para 27 linguagens de programação, incluindo: C, C#, Java, Javascript e Python [4]. Destaca-se que a base de regras é evoluída continuamente pela SonarSource[5], empresa detentora dos direitos sobre o SonarQube[6]. É importante mencionar que as regras implementadas estão associadas a um nível de gravidade, a saber:

- *Blocker*: trata-se de um problema com alta probabilidade de impactar o comportamento da aplicação em produção. Ex: vazamento de memória, manutenção de conexões ao banco de dados abertas, etc.. Recomenda-se que problemas associados a regras com este nível de gravidade precisam ser corrigidos imediatamente [3];
- *Critical*: refere-se a um problema com baixa probabilidade de impactar o funcionamento da aplicação ou uma provável brecha de segurança. Ex: blocos de tratamento de exceção vazios, possibilidade de injeção de SQL, etc.. É recomendado que o código nessa situação seja verificado o quanto antes [3];
- *Major*: trata-se de um problema de qualidade do código com grandes chances de impactar a produtividade do time de desenvolvimento. Ex: blocos de código duplicados, parâmetros inutilizados, etc.;
- *Minor*: refere-se a problemas de qualidade do código com menor probabilidade de impactar o desenvolvimento e manutenção da aplicação. Ex.: linhas de código muito extensas, declarações *switch* com menos de 3 possibilidades, etc.;
- *Info*: não se trata nem de um *bug* nem de um problema de qualidade mas apenas um aviso de que algo deve ser verificado.

Uma organização que deseja utilizar o SonarQube na avaliação do código-fonte pode incorporá-lo ao processo de desenvolvimento de duas maneiras: *on premise* ou *cloud*. No modelo *on premise* deve-se instalar a ferramenta em um servidor da organização, sendo de responsabilidade da própria empresa o provimento de todos os recursos de infraestrutura necessários. Em contrapartida, para a utilização do modelo *cloud*, basta criar uma conta na SonarCloud[7]. É possível utilizar gratuitamente o modelo *cloud* porém, para isto, é exigido que a análise do código fique disponível publicamente, isto é, empresas que desejam utilizar a ferramenta na nuvem, de forma privada, precisam pagar por isto.

Diante deste contexto, neste trabalho, pretende-se realizar um estudo acerca dos problemas encontrados nos projetos com análise pública disponíveis na SonarCloud[8]. Atualmente existem 207.769 projetos com análise pública disponível. Sabe-se que o conjunto de regras implementado para cada linguagem é diferente,

[4] A lista de todas as linguagens com regras implementadas está disponível em https://www.sonarqube .org/features/multi-languages

[5] https://www.sonarsource.com

[6] A lista de todas as regras implementadas, por linguagem, se encontra disponível em https://rules.sonarsource.com

[7] https://sonarcloud.io

[8] Os projetos disnponíveis publicamente podem ser visualizados em https://sonarcloud.io/explore/projects

pela particularidade de alguns recursos que a respectiva linguagem disponibiliza bem como por efeito de priorização da SonarSource na impementação de novas regras. Portanto, definiu-se como escopo deste trabalho os projetos escritos na linguagem C# (18072) [9] com número de linhas superior a 50.000 [10]. Após a realização deste filtro o número de projetos considerados passou a ser de 4.216. Para a linguagem C#, objeto de estudo deste trabalho, existem, atualmente, 381 regras implementadas com status ativo sendo 76 referentes à *Bugs*, 275 para a identificação de *Code Smells* e 30 relacionadas à Vulnerabilidades.

2 Metodologia

A metodologia adotada neste trabalho pode ser dividida em três etapas, a saber: Extração, Tratamento e Análise dos Resultados. Nas próximas subseções serão apresentados detalhes acerca de cada etapa mencionada. Todo o código criado para automação das etapas se encontra disponível no Github. Após a execução dessas etapas, espera-se identificar:

- A proporção individual de *Code Smells*, *Bugs* e Vulnerabilidades relatados nos projetos, por nível de gravidade;
- Os *Code smells*, *Bugs* e Vulnerabilidades com o maior número de ocorrências, contabilizados a partir do somatório das ocorrências em todos os projeto considerados (análise de recorrências);
- Os *Code smells*, *Bugs* e Vulnerabilidades reportados individualmente no maior número de projetos (análise de ocorrências);

2.1 Extração

Os dados utilizados nos experimentos foram extraídos da API pública da SonarCloud[11]. Abaixo detalha-se os recursos/métodos utilizados juntamente com os respectivos parâmetros considerados na montagem do filtro:

- **GET /api/rules/search:** Lista as regras disponíveis.
  - ***organization*:** Chave única que identifica a organização para a qual os projetos serão extraídos. Nos experimentos é considerado o valor *default* utilizado para a listagem de todos os projetos públicos. Valor: *explore*;
  - ***languages*:** Linguagem de programação (C#). Valor: *cs*;
  - ***statues*:** Apenas regras ativas, não depreciadas. Valor: *READY*;
  - ***types*:** Os três tipos de problemas, escopo deste trabalho. Valor: *CODE_SMELL,BUG,VULNERABILITY*.

[9] A linguagem C# foi escolhida pois é aquela que o autor possui mais domínio, com maior capacidade de análise.

[10] Entende-se que quanto maior o número de linhas, menores as chances de que o projeto seja experimental (prova de conceito, estudos, etc.)

[11] O endereço base da API é https://sonarcloud.io/api. Sua documentação pode ser encontrada em https://sonarcloud.io/web_api.

- **GET /api/components/search_projects:** Lista os projetos disponíveis de acordo com os filtros informados.
  - ***filter*:** mecanismo utilizado pela API para filtrar os projetos listados por tamanho e linguagem. Valor: *nloc* $\geq$ *50000 and languages = cs*.
- **GET /api/measures/component:** Para cada projeto retornado pelo método *GET /api/components/search_projects*, apresenta o valor das métricas desejadas.
  - ***component*:** identificador do projeto. Valor: *<project-key>*;
  - ***metricKeys*:** representam, respectivamente, o número de linhas, número de linhas por linguagem, número de classes e percentual de linhas cobertas por teste de unidade. Valor: *ncloc,lcloc_language,classes,line_coverage*.
- **GET /api/issues/search:** Lista os problemas encontrados em determinado projeto.
  - ***componentKey*:** identificador do projeto. Valor: *<project-key>*;
  - ***types*:** os três tipos de problemas de codificação, escopo deste trabalho. Valor: *CODE_SMELL,BUG,VULNERABILITY*;
  - ***languages*:** linguagem C#. Valor: *cs*;
  - ***statues*:** apenas problemas não tratados. Valor: *OPEN,CONFIRMED, REOPENED*.

Para que se tenha uma melhor compreensão, um exemplo dos dados retornados como resposta na chamada de cada API pode ser encontrado no caminho *data/api/response_examples* do diretório anexo. Destaca-se que o processo de extração foi automatizado por meio da construção de um *script* desenvolvido na linguagem Python com o auxílio das bibliotecas *requests* e *json*.

2.2 Tratamento

Na fase de tratamento fez-se necessária a realização de um novo filtro sobre os 4.216 projetos considerados inicialmente. Isso se deve ao fato de que muitos dos projetos retornados possuem nome, quantidade de linhas de código, número de *Bugs*, *Code Smells* e Vulnerabilidades idênticos. Deste modo, projetos que possuem em seu nome as palavras "part" e "unlimited", identificadas como ofensoras neste sentido, foram desconsiderados, o que possibilitou uma redução do número de projetos analisados para 1.907 (-54,77%). Para que se tenha uma dimensão do conjunto de dados resultante, nestes projetos, o número total de linhas de código escritas em C# é igual a 47.455.971 e o número de classes é de 696.676.

Os dados extraídos possuem algumas particularidades que dificultariam a análise caso fossem considerados de forma bruta. A Figura 1 ilustra este fato, a qual apresenta o retorno do método *GET /api/measures/component*. Nessa figura é possível perceber que, para a métrica *ncloc_language_distribution*, que representa o número de linhas de código por linguagem de programação existente no projeto, os dados são apresentados no formato chave/valor em uma única *string*, fazendo-se necessária a realização de um tratamento. Neste caso, o tratamento realizado foi a transformação dos dados em uma lista.

Após o tratamento dos dados, foram criadas três entidades, a saber:

Fig. 1: Retorno do método *GET /api/measures/component*.

- ***Rule:*** Representa uma regra
  - ***Key*:** Chave única para identificação da regra;
  - ***Name*:** Nome da regra;
  - ***Severity*:** Gravidade da regra (INFO, MINOR, MAJOR, CRITICAL, BLOCKER);
  - ***Type*:** Tipo da regra (*BUG*, *CODE_SMELL* ou *VULNERABILITY*).
- ***Issue:*** Representa um problema, podendo ser um *Code Smell*, *Bug* ou Vulnerabilidade.
  - ***rule*:** Chave da regra apontada;
  - ***message*:** Mensagem com a descrição do problema.
- ***Project:*** Representa o projeto ou repositório
  - ***bugs*:** Lista de *Bugs* (*Issue* do tipo *BUG*) encontrada no projeto;
  - ***code_smells*:** Lista de *Code Smells* (*Issue* do tipo *Code Smell*) encontrada no projeto;
  - ***metric_lines_of_code*:** Número de linhas de código;
  - ***metrics_lines_of_code_per_language*:** Número de linhas de código por linguagem utilizada no projeto;
  - ***metrics_classes*:** Número de classes;
  - ***metrics_line_coverage*:** Percentual de linhas cobertas por testes de unidade;
  - ***project_key*:** Identificador único do projeto;
  - ***project_name*:** Nome do projeto;
  - ***vulnerabilities*:** Lista de Vulnerabilidades (*Issue* do tipo *VULNERABILITY*) encontrada no projeto.

Os dados extraídos foram salvos nos arquivos com extensão *.dat*, que podem ser visualizados no diretório */data*. O arquivo *rules.dat* contém a lista de regras (entidade *Rule*) para a linguagem C# e o arquivo *projects.dat* possui a lista de projetos (entidade *Project*) considerados na análise.

2.3 Análise dos Resultados

A Figura 2 apresenta a proporção de *Bugs*, *Code Smells* e Vulnerabilidades, por nível de gravidade, nos projetos analisados. Percebe-se que para problemas dos tipos *Bugs* e *Code Smells* as regras com maior número de violações são de gravidade MAJOR (índice 3 em uma escala de 1 a 5). Ainda para estas categorias, as violações de regras com gravidade BLOCKER representam menos de 3% dos problemas relatados. No que se refere à problemas de Vulnerabilidade, regras de gravidade MINOR (2 em 5) foram as mais violadas. Neste caso, quase 10% dos problemas relatados têm o nível de gravidade máximo (BLOCKER).

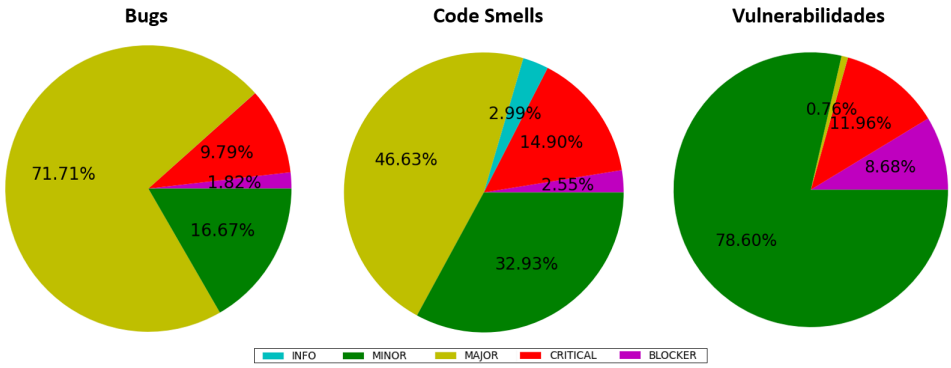

Fig. 2: Proporção de *Bugs*, *Code Smells* e Vulnerabilidades por nível de gravidade.

Sabe-se que problemas com gravidade BLOCKER são os que geram maiores preocupações. Nesse sentido, com o objetivo de aprofundar nos problemas mais recorrentes com este nível de gravidade, foram realizadas duas análises:

- Recorrência: verifica as regras com o maior número de violações, contabilizando todas as recorrências da respectiva regra, em todos os projetos;
- Ocorrência: neste caso, cada projeto é contabilizado uma única vez, independente do número de violações encontradas no projeto para a regra analisada.

As Tabelas 1 e 2 apresentam um resumo dessa análise e contém as 3 regras mais violadas referentes, individualmente, a *Bugs* (B), *Code Smells* (C) e Vulnerabilidades (V), sob essas duas perspectivas.

Ao analisar a Tabela 1, percebe-se que as regras S1944 (B) e S2223 (C) são muito recorrentes, porém, a não existência dessas regras na Tabela 2 demonstra

Table 1: Análise por Recorrência: regras com o maior número de violações

Tipo	Id	Nome	#
B	S2551	Shared resources should not be used for locking.	1695
	S4275	Getters and setters should access the expected fields.	1038
	S1944	Inappropriate casts should not be made.	64
C	S3776	Cognitive Complexity of methods should not be too high.	44.156
	S927	Parameter names should match base declaration and other partial definitions.	18.308
	S2223	Non-constant static fields should not be visible.	11.294
V	S2486	Generic exceptions should not be ignored.	2.648
	S5547	Cipher algorithms should be robust.	138
	S3330	Creating cookies without the "HttpOnly" flag is security-sensitive	132

Table 2: Análise por Ocorrência: regras com o maior número de violações

Tipo	Id	Nome	#
B	S4275	Getters and setters should access the expected fields.	137
	S2551	Shared resources should not be used for locking.	79
	S4586	Non-async "Task/Task<T>" methods should not return null.	23
C	S3776	Cognitive Complexity of methods should not be too high.	1.764
	S1186	Methods should not be empty.	1.680
	S927	Parameter names should match base declaration and other partial definitions.	1.679
V	S3330	Creating cookies without the "HttpOnly" flag is security-sensitive	114
	S2486	Generic exceptions should not be ignored.	59
	S5547	Cipher algorithms should be robust.	42

que elas são reportadas em um número menor de projetos, sendo substituídas nessa tabela pelas regras S4586 (B) e S1186 (C), respectivamente. Para o entendimento da importância dessa análise nos projetos de uma organização, os problemas que podem ser causados pela sua ocorrência e as possibilidades de correção, recomenda-se a leitura da documentação da base de regras implementadas, disponível em [2].

3 Considerações Finais

Este trabalho apresentou uma análise preliminar da qualidade do código-fonte de projetos públicos disponíveis na SonarCloud. Nesta análise foram considerados os problemas reportados de forma automatizada pela ferramenta referentes a *Bugs*, *Code Smells* e Vulnerabilidades . De forma geral, o objetivo do trabalho consistiu em estabelecer uma proposta de metodologia para extração, tratamento e visualização dos problemas reportados pelo SonarQube/SonarCloud de forma

a contribuir na qualidade do código-fonte produzido por um time de desenvolvimento.

Sugere-se como trabalho futuro a aplicação dessa metodologia em uma empresa com projetos reais, identificando os principais problemas e estabelecendo ações que visem coibí-los nos projetos da organização. Além disso, outras métricas também disponibilizadas pelo SonarQube/SonarCloud podem ser analisadas, tais como o índice de cobertura do código por testes de unidade, *security hotspots*, dentre outros.

References

1. Pressman, R.: Engenharia de Software: uma abordagem profissional. Bookman, Porto Alegre, RS, Brasil (2011)
2. SonarCloud: Rules Documentation (2021), acessado em: 08/10/2021
3. SonarSource: Issue Types (2021), acessado em: 07/10/2021