Analisando Estratégias de Identificação de Dívidas Técnicas

Isabela Oliveira[1], **Humberto T. Marques-Neto**[1], **Laerte Xavier**[1]

[1] Departamento de Engenharia de Software e Sistemas da Informação (DES)
Pontifícia Universidade Católica de Minas Gerais (PUC-Minas)
Belo Horizonte – MG – Brasil

isabela.edilene@sga.pucminas.br

humberto@pucminas.br, jlpxjunior@sga.pucminas.br

Abstract. *Technical Debt (TD) is a constant in software development. To deal with the pressure of fast deliveries, developers constantly compromise the quality of the delivered system, postponing TD payment to later stages. In order to reduce its negative effects, several solutions are proposed in the literature for the identification, management and payment of these debts. Among them, we highlight the approaches based on static analysis (code smells), or on code comments (self-admitted technical debt, or SATD). However, the real intersection between such approaches is not yet clear, nor are the characteristics of the debts best identified by each of them. In this work, we perform a comparison between both approaches through the application of SonarQube and SATDDetector in a dataset of 1,000 popular GitHub repositories. As a result, we observe that the intersection between them is approximately 19%, and that in 7% of the cases SonarQube is not able to identify SATDs. In addition, the debts identified by both approaches are related to larger files (in terms of lines of code), more complex (cyclomatic and cognitive complexity) and more smelly (code smells).*

Resumo. *Dívidas técnicas são uma constante no desenvolvimento de software. Para lidar com a pressão de entregas rápidas, desenvolvedores constantemente comprometem a qualidade do sistema entregue, postergando seu pagamento para fases posteriores. Com o objetivo de reduzir seus efeitos negativos, diversas soluções são propostas na literatura para identificação, gerenciamento e pagamento destas dívidas. Dentre elas, destacam-se as abordagens baseadas em análise estática (code smells), ou em comentários de código (self-admitted technical debt, ou SATD). Entretanto, ainda não está clara a real interseção entre tais abordagens, tampouco as características das dívidas melhor identificadas por cada uma delas. Neste trabalho, realizou-se uma comparação entre ambas abordagens através da aplicação das ferramentas SonarQube e SATD-Detector num conjunto composto por 1.000 repositórios populares do GitHub. Como resultado, verificou-se que a interseção entre elas é de aproximadamente 19%, e que, em 7%, dos casos o SonarQube não é capaz de identificar SATDs. Ademais, as dívidas identificadas por ambas abordagens estão relacionadas à arquivos maiores (em termos de linhas de código), mais complexos (complexidade ciclomática e cognitiva) e com mais code smells.*

1. Introdução

Espera-se que o desenvolvimento de um software alcance um código de qualidade, uniforme e compreensível. Entretanto, um fenômeno recorrente neste processo é denominado *dívida técnica (DT)*, uma metáfora utilizada para descrever as situações em que a qualidade do código é negociada para que objetivos de curto prazo sejam priorizados [Huang et al. 2018]. Nesse contexto, o termo *Self-Admitted Technical Debt* (SATD) identifica situações em que, intencionalmente, os desenvolvedores introduzem dívida técnica no código e o documentam através de comentários [Potdar and Shihab 2014].

Diversas abordagens são propostas com o objetivo de identificar dívidas técnicas [Xavier et al. 2020, Maldonado et al. 2017, Potdar and Shihab 2014]. Porém, permanece em aberto o questionamento acerca da equivalência do entendimento dessas abordagens com relação às dívidas admitidas por desenvolvedores. Portanto, **este trabalho tem como objetivo analisar a interseção entre abordagens de identificação de declarações de dívida técnica direto em códigos fonte e a abordagem de identificação automatizada de dívidas técnicas em ferramentas como SonarQube**. Assim, deseja-se ao final deste trabalho responder as seguintes questões de pesquisa:

- **RQ1:** *Qual é a interseção entre as abordagens de identificação automática de dívida técnica e os SATDs?*
- **RQ2:** *Quais são as características dos arquivos que possuem DTs identificadas por cada abordagem?*

A fim de investigar tais perguntas, será utilizada a ferramenta SonarQube para a abordagem de identificação automatizada de dívidas técnicas. Trata-se de uma ferramenta *open-source*, utilizada para analisar e gerenciar a qualidade de código estático, que identifica dívida técnica por meio da análise de *code smells*. Já para coletar as dívidas técnicas que foram declaradas explicitamente pelos próprios desenvolvedores (SATDs), será utilizado o SATDDetector, um *plug-in* desenvolvido por Liu et al. [Liu et al. 2018]. Essa ferramenta faz uso de aprendizado de máquina para identificar comentários no código que possam indicar uma dívida técnica.

Estrutura do artigo: a Seção 2 apresenta a metodologia aplicada. Em seguida, apresenta-se os resultados obtidos na Seção 3, e as discussões acerca dos mesmos na Seção 4. As ações tomadas para minimizar algumas ameaças à validade são apresentadas na Seção 5. Por fim, as Seções 6 e 7 discutem os trabalhos relacionados e as conclusões obtidas.

2. Metodologia

A fim de responder às questões de pesquisa propostas, foi minerado um *dataset* composto por repositórios de projetos *open-source* hospedados no GitHub. Para tanto, foram selecionados 1.000 repositórios populares da linguagem Python, baseando-se (i) no número de estrelas de cada repositório [Borges and Valente 2018]; e (ii) na ocorrência de pelo menos uma dívida identificado pelo SADTDetector. Isto é, os 1.000 repositórios Python mais populares, com pelo menos um SATD identificado. O processo de obtenção dos dados e dos resultados finais foi dividido nas seguintes etapas:

Seleção do *dataset* com o SATDDetector. Para a realização da coleta, tratamento e análise parcial dos dados, foi desenvolvido um programa para clonar cada um dos repositórios, identificar e extrair todos os comentários dos arquivos com extensão ".py" do

projeto. Em seguida, cada comentário foi analisado pelo SATDDetector, que identifica e salva se o comentário analisado representa um SATD. Finalmente, foram geradas duas bases para análises posteriores: uma primeira contendo os 1.000 repositórios mais populares que tiveram pelo menos um SATD identificado em todo o projeto; e uma segunda contendo a identificação dos *arquivos* do projeto que possuem SATD. Os repositórios que não possuem nenhum SATD não foram considerados neste trabalho.

Análise com o SonarQube. Para análise dos projetos com o SonarQube, foi desenvolvido um *script* responsável por (i) clonar todos os repositórios do *dataset*, nos quais foram identificados pelo menos um SATD; e (ii) criar um novo projeto no SonarQube, importando o repositório clonado do GitHub. Após a execução da ferramenta, foi uma produzida uma terceira base, contendo a identificação dos arquivos de cada projeto que possuem dívida técnica, de acordo com o SonarQube.

Caracterização das dívidas técnicas. Por fim, as instâncias identificadas em cada uma das abordagens foram agrupadas em três grupos:

- **Grupo 1 (G1):** dívidas identificadas pela abordagem automatizada e admitidas pelos desenvolvedores. Isto é, dívidas identificadas por ambas abordagens.
- **Grupo 2 (G2):** dívidas admitidas pelos desenvolvedores. Isto é, identificadas somente pelo SATDDetector.
- **Grupo 3 (G3):** dívidas identificadas somente pela abordagem automatizada. Isto é, identificadas somente pelo SonarQube.

Para cada um dos grupos, foram coletadas as seguintes métricas, a nível de arquivo: número de *Code smells*; NcLOC; complexidade ciclomática; e complexidade cognitiva. Os *scripts* desenvolvidos para coleta e análise dos projetos, bem como os dados obtidos neste estudo, estão disponíveis publicamente[1].

3. Resultados

RQ1: *Qual é a interseção entre as abordagens de identificação de dívida técnica?*

Após clonar os primeiros 1.000 repositórios selecionados nesse estudo (Seção 2), foram analisados 169.259 arquivos, dos quais 67.188 registraram alguma dívida técnica (DT) de acordo com uma das abordagens. A Figura 1 detalha o total de arquivos em que foram identificadas pelo menos uma dívida técnica, bem como a divisão dos mesmos de acordo com a classificação dos grupos definidos na Seção 2. No **G1** (ambas abordagens), um total de 13.082 arquivos (19,47%) incluem instâncias de dívidas identificados pelas duas abordagens. Já os grupos **G2** (SATDDetector) e **G3** (SonarQube) possuem um total de 4.677 (6,96%) e 49.429 (73,57%) arquivos, respectivamente.

Com o objetivo de ilustrar os resultados deste trabalho, realizou-se uma análise manual do projeto Django. Para este projeto, o grupo **G1** (ambas abordagens) inclui instâncias de dívidas documentadas pelos desenvolvedores da seguinte maneira: *"TODO: Handle multiple backends with different feature flags"*, *"Move positional args out of options to mimic legacy optparse"* e *"could create superusers, which would mean they would essentially have"*

No grupo **G2** (SATDDetector), observa-se que os comentários deixados pelo desenvolvedores são relacionados à transações de banco de dados, por exemplo *"blocks.*

[1] `https://github.com/isabelaedilene/technicalDebtTisVI`

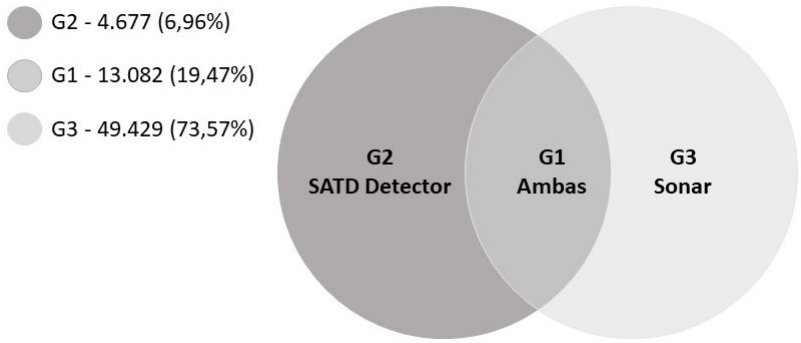

Figura 1. Porcentagem da identificação de arquivos com dívidas técnicas em relação às abordagens utilizadas.

To work around that bug, on SQLite, 'atomic' starts a...", *"SQLite-ish workaround"* e *"UNION operator. To prevent incorrect SQL, move UNION to a subquery."*. Neste grupo, são registrados também comentários relacionados à dúvidas e lembretes, tais *"What kind of error does the backend throw when accessing closed cursor?"* e *"See if any of the handlers take care of the parsing."*. Por fim, encontram-se nesse grupo comentários relacionados ao funcionamento do código que indicam que algo já poderia ter sido realizado, mas foi deixado para outro momento: *"TODO: Remove str() when dropping support for PY37"* e *"FIXME: this currently assumes that upload handlers store the file as file"*.

No grupo **G3** (SonarQube), observa-se que os comentários dos arquivos são, em geral, relacionados à documentação do código, como *"Test atomic operation in non-atomic migration is wrapped in transaction"*. Em muitos casos, os arquivos analisados não possuem nenhum comentário.

RQ2: *Quais são as características dos arquivos que possuem DTs identificadas por cada abordagem?*

Além da porcentagem de dívidas técnicas identificadas em cada abordagem, foram analisadas algumas características dos arquivos incluídos em cada um dos grupos, conforme apresentado na Figura 2.

O grupo **G1** (dívidas identificadas por ambas abordagens) apresenta arquivos maiores, ou seja, com mais linhas de código em relação aos demais. **G1** possui um valor mediano de 264 linhas (sem considerar comentários), enquanto o **G2** possui 72 e **G3** 107 linhas. Em relação à complexidade ciclomática e cognitiva, o grupo **G1** também apresenta valores maiores em relação aos demais. A complexidade ciclomática dos grupos possuem um valor mediano de 48 em **G1**, 11 em **G2** e 16 em **G3**. Já a cognitiva 45 em **G1**, 7 em **G2** e 11 em **G3**. No caso do valor mediano de *code smells*, a característica não se aplica ao grupo **G2**, já que esse grupo agrupa arquivos que não tiveram nenhuma dívida técnica identificada pelo SonarQube, e por isso, o seu valor está zerado no gráfico. Já o grupo **G1** possui valor mediano de 4 *code smells* enquanto o grupo **G3** possui 2 *code smells* como valor mediano.

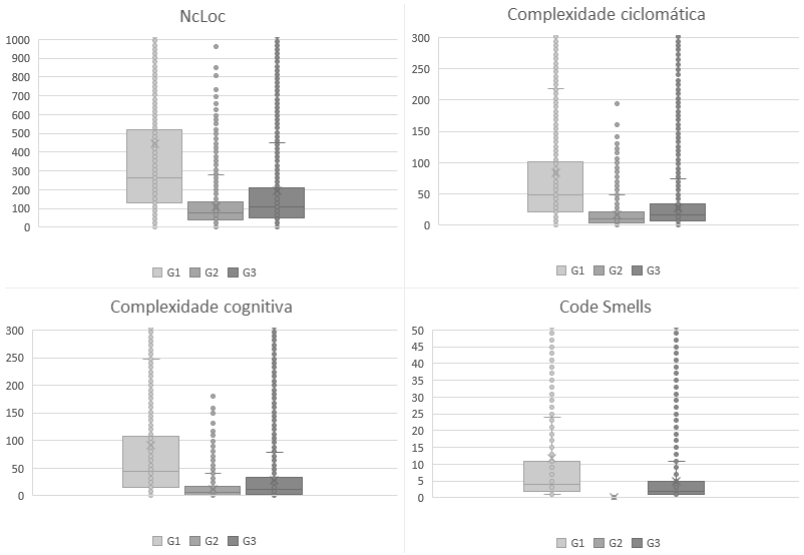

Figura 2. Métricas analisadas dos grupos formados com base na identificação por abordagem.

Em resumo, pode-se observar que o grupo **G1** possui os valores mais altos para todas as métricas avaliadas, enquanto o grupo **G2** possui os mais baixos.

4. Discussão

Após analisar os resultados apresentados na Seção 3, algumas percepções e conclusões podem ser obtidas com os mesmos.

Zazworka et al. descrevem que, para os considerados *dívidas de defeito*, as abordagens automatizadas conseguiram identificar praticamente todos os itens que foram identificados de forma manual [Zazworka et al. 2013]. Entretanto, apenas metade das *dívidas de design* identificados de forma manual foram identificados pelas abordagens automatizadas. Os autores concluíram que as abordagens automatizadas são úteis para identificação de dívidas de defeito, o que é semelhante ao resultado encontrado no presente trabalho, se considerarmos que o SonarQube foi capaz de identificar um total de 73,57% de dívida técnica. Isso significa que há a possibilidade do grupo dos SATDs não encontrados ter características de dívidas de *design*.

As demais dívidas que não foram identificadas pelo SonarQube também podem ser tipos de dívidas que não são consideradas ou desconhecidas. No estudo de Zazworka et al., os autores identificaram que, dos 21 itens analisados, apenas 1 foi relatado por dois membros da equipe. Nenhum, dos outros 19 itens, foi relatado [Zazworka et al. 2013]. De acordo com os autores, isso mostra que o conhecimento de dívida técnica, naquele projeto analisado, é disperso e percebido de maneira diferente por diferentes membros da equipe. O *tester* da equipe foi o que relatou mais tipos de dívidas técnicas, inclusive tipos desconhecidos, como dívida de usabilidade.

Além disso, para esse estudo, foram considerados os arquivos dos projetos analisados, e não especificamente ao método/função que tiveram a dívida técnica admitida. Essa taxa pode sofrer alterações ao se realizar uma análise mais detalhada. Porém, é importante observar que existe a possibilidade de uma dívida técnica estar distribuída em

muitos arquivos, ou estar relacionada à outros artefatos do desenvolvimento de software, como documentos de requisitos, plano de testes, uso de recursos do sistema, etc., o que significa que elas também poderiam não ser capturadas pelas abordagens automatizadas.

Deve-se observar também, que a quantidade de arquivos em cada grupo são bem distantes entre si. Isso significa que os valores das métricas (linhas de código, complexidade ciclomática e cognitiva) coletadas para os grupos também poderão ser distantes uns dos outros.

5. Ameaças à validade

Para minimizar algumas ameaças à validade, as seguintes ações foram tomadas:

Validade da conclusão. Obtivemos uma amostra de dados relevante (1.000 projetos). Além disso, o relacionamento considerado é apenas o número de dívidas identificadas por cada abordagem, a fim de não criar correlações falsas entre variáveis.

Validade do construto. Em relação ao momento em que as medidas são obtidas, é necessário que os projetos sejam analisados nas mesmas condições, ou seja, o projeto deve ser clonado em um momento específico para avaliação nas duas ferramentas. Não deve-se cloná-lo, avaliá-lo em uma das ferramentas, e depois cloná-lo novamente para a avaliação em outra ferramenta. Isso poderia trazer mudanças no projeto. Além disso, deve-se observar que as dívidas inicialmente coletadas são admitidas pelos desenvolvedores. Os resultados obtidos pelo Sonar são confrontados com esses valores, sem que haja uma validação posterior (que está fora do escopo do trabalho).

Validade interna. As variáveis independentes afetariam as dependentes apenas se ao invés de usar a combinação SATDDetector e SonarQube, outro conjunto de ferramentas fosse escolhido. Haveria a relação esperada, mas o número total de SATD identificados poderia ser outro. Além disso, uma outra versão do software poderia oferecer uma eficácia maior, afetaria também o número de *code smells* encontrados. Para a identificação dos SATDs com o SATDDetector, foi usado o modelo padrão que a ferramenta disponibiliza para realização de testes. O uso de modelos que incluíssem outras línguas além do inglês, certamente resultaria em conclusões diferentes.

Validade externa. Considerando que o SonarQube é uma ferramenta já validada e muito utilizada em ambientes acadêmicos e no mercado profissional, acredita-se que os seus resultados são confiáveis e portanto podem ser generalizados para qualquer ambiente em que ele seja utilizado. Entretanto, não se pode generalizar as observações apresentadas para outros contextos de desenvolvimento, tampouco outros domínios de software.

6. Trabalhos Relacionados

Code Smells indicam possíveis problemas na implementação de um sistema e são indicadores de presença de dívidas técnicas [Gomes et al. 2019]. Considerando que o SonarQube define dívida técnica como "O tempo estimado necessário para corrigir todos os problemas de manutenção/*code smells*" e é necessário uma métrica que indique a quantidade de dívida técnica para este estudo, deve-se utilizar a métrica *Code Smell* da ferramenta SonarQube. De acordo com as regras do SonarQube, *code smells* são considerados problemas de manutenção. Para Fowler et al., *code smells* são sintomas superficiais de um problema mais profundo no seu código. Assim, ele define um total de 22 tipos específicos de *code smells* [Fowler et al. 1999]. Desses, o SonarQube considera apenas

4, que são: código duplicado, método longo, classe grande e lista longa de parâmetros [Lenarduzzi et al. 2020]. O SonarQube disponibiliza em sua documentação oficial as regras utilizadas para identificação de *code smells*. É possível ver um exemplo sobre o que é esperado e uma classificação do tipo e severidade do *code smell*.

Lenarduzzi et al. concluíram que as regras utilizadas pelo SonarQube para calcular dívida técnica devem ser investigadas e a sua nocividade também precisa ser confirmada. Assim, as empresas devem observar quais regras desejam aplicar em seus projetos, principalmente se seu objetivo é reduzir a propensão a falhas [Lenarduzzi et al. 2020].

Zazworka et al. conduziram um estudo semelhante ao presente trabalho com foco na identificação da dívida técnica [Zazworka et al. 2013]. Para isso, foi realizado uma elicitação humana de dívidas técnicas, ou seja, os autores criaram um *template*, semelhante à uma lista de *backlog*, e solicitaram que um time de desenvolvimento reportasse itens de dívida técnica individualmente. Paralelamente, os autores aplicaram as ferramentas CodeVizard e FindBugs na versão mais recente do código-fonte do projeto em questão. Os autores concluíram com essa pesquisa que as ferramentas testadas são úteis para identificar dívidas técnicas de defeito, mas não podem ajudar na identificação de outros tipos de dívida. Assim, os autores acreditam ser necessário envolver a identificação manual, e não apenas ferramental, nesse processo.

No trabalho de Potdar et.al [Potdar and Shihab 2014], os autores evidenciaram que apenas entre 26,3 e 63,5% do SATD introduzido é removido posteriormente. Inspirados pelo trabalho de [Huang et al. 2018], [Liu et al. 2018] desenvolveram uma ferramenta que consiste em uma biblioteca Java como *back-end* e um *plug-in* para a IDE Eclipse como *front-end*. Através de uma biblioteca Java, os usuários treinam modelos de mineração de textos, via *machine learning*, de forma a personalizar/aprimorar a detecção de SATD. A ferramenta pode ser utilizada no auxílio ao desenvolvedor na identificação e manutenção de SATDs.

7. Conclusão

O estudo realizado nos permitiu analisar e comparar se diferentes abordagens de identificação de dívida técnica possuem o mesmo entendimento sobre o problema investigado. Como resultado, observou-se que 19,47% das dívidas técnicas analisadas são admitidas por desenvolvedores e também identificadas pelo Sonar. Isto é, possivelmente elas possuem um entendimento complementar. Infelizmente, não foi encontrado na literatura nenhum valor relacionado à eficácia ou à taxa de identificação que nos ajudasse a concluir se esse número representa um valor positivo ou se alcança algum valor desejado. Entretanto, estudos relacionados possuem conclusões semelhantes em relação à diferença de identificação de dívidas técnicas que pode ser obtida em diferentes abordagens.

Os resultados apresentados neste trabalho podem ser utilizados para estudos e pesquisas que visam realizar uma comparação entre abordagens de identificação de dívida técnica. Além disso, desenvolvedores podem se beneficiar no processo de escolha de alguma abordagem para auxiliar no pagamento de suas dívidas. Como trabalho futuro, este estudo pode evoluir e aprofundar mais questões não abordadas: analisar outras características dos grupos formados, refinar a metodologia para uma análise a nível de funções do código, e utilizar outras ferramentas e abordagens para identificação de dívida técnica.

Referências

[Borges and Valente 2018] Borges, H. and Valente, M. T. (2018). What's in a GitHub star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129.

[Fowler et al. 1999] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). Refactoring: Improving the design of existing code. *Berkeley, CA, USA*.

[Gomes et al. 2019] Gomes, F. G. d. S., Mendes, T. S., Spínola, R. O., Mendonça, M., and Farias, M. (2019). Uma análise da relação entre code smells e dívida técnica auto-admitida. In *7th Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pages 37–44.

[Huang et al. 2018] Huang, Q., Shihab, E., Xia, X., Lo, D., and Li, S. (2018). Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering*, 23(1):418–451.

[Lenarduzzi et al. 2020] Lenarduzzi, V., Lomio, F., Huttunen, H., and Taibi, D. (2020). Are sonarqube rules inducing bugs? In *27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 501–511.

[Liu et al. 2018] Liu, Z., Huang, Q., Xia, X., Shihab, E., Lo, D., and Li, S. (2018). SATD Detector: A text-mining-based self-admitted technical debt detection tool. In *40th International Conference on Software Engineering (ICSE)*, pages 9–12.

[Maldonado et al. 2017] Maldonado, E. d. S., Abdalkareem, R., Shihab, E., and Serebrenik, A. (2017). An empirical study on the removal of self-admitted technical debt. In *33th International Conference on Software Maintenance and Evolution (ICSME)*, pages 238–248.

[Potdar and Shihab 2014] Potdar, A. and Shihab, E. (2014). An exploratory study on self-admitted technical debt. In *30th International Conference on Software Maintenance and Evolution (ICSM)*, pages 91–100.

[Xavier et al. 2020] Xavier, L., Ferreira, F., Brito, R., and Valente, M. T. (2020). Beyond the code: Mining self-admitted technical debt in issue tracker systems. *17th International Conference on Mining Software Repositories (MSR)*.

[Zazworka et al. 2013] Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., and Seaman, C. (2013). A case study on effectively identifying technical debt. In *17th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 42–47.