DOI: 10.36558/rsc.v13i1.7991

AVALIAÇÃO DO USO DE HEURÍSTICAS DE CONTEXTO NA MINERAÇÃO DE DÍVIDA TÉCNICA COM A EXCOMMENT

AN ASSESSMENT OF THE USE OF CONTEXT HEURISTICS IN THE TECHNICAL DEBT MINING WITH EXCOMMENT

Diogo Alves de Moura Loiola
Universidade Estadual do Piauí – UESPI
Piripiri - PI
Brasil
diogoloiola@aluno.uespi.br

Tchalisson Brenne Santos Gomes
Universidade Estadual do Piauí – UESPI
Piripiri – PI
Brasil
tchalissongomes@aluno.uespi.br

Alcemir Rodrigues Santos
Universidade Estadual do Piauí – UESPI
Piripiri – PI
Brasil
alcemir@prp.uespi.br

Mário André de Freitas Farias
Instituto Federal do Sergipe – IFS
Lagarto – SE
Brasil
mario.andre@ifs.edu.br

RESUMO

Contexto: A eXcomment é uma ferramenta que minera comentários em código-fonte para identificar itens de Dívida técnica (DT) de maneira automatizada. *Problema:* Após estudos preliminares com a primeira versão, os autores introduziram heurísticas no processo de identificação de DT, o que produziu a segunda versão. Até este ponto, não existia um estudo empírico para avaliar como o uso de heurísticas pode afetar na identificação de DT pela ferramenta. *Método:* Este artigo investiga o efeito destas heurísticas no resultado da mineração de DT através de dois estudos de caso com softwares de código aberto. *Resultados:* Os resultados mostraram redução na quantidade de falsos positivos, o que levou melhoria à da bem como foi possível perceber ligeira melhroa na *acurácia* da ferramenta.

Palavras-Chave

Dívida Técnica, Avaliação; eXComment; Heurísticas de Contexto.

ABSTRACT

Context: The eXcomment is a tool built to find technical debt through the mining of source code comments. *Problem:* After preliminary evaluation of the first version of the tool, the authors added additional heuristics to improve its results, which resulted in a second version of it. So far, no evaluation has been done on the effect of such heuristics on the technical debt identification. *Method:* Therefore, this paper evaluates the effect of such heuristics on the mining by the tool with two open-source use cases. *Results:* Results show fewer false positives, which led to increased *accuracy*.

Keywords

Technical Debt; Evaluation; eXComment; Context Heuristics

1. INTRODUÇÃO

Desenvolvedores estão sob constante pressão, para prover softwares úteis e que tenham alta qualidade de código. Desenvolver um software que esteja de acordo com os requisitos planejados é o cenário ideal, mas nem sempre é o que acontece. Fatores externos podem afetar de maneira significativa esse processo, fazendo que os desenvolvedores tomem de atalhos para concluir suas atividades de maneira mais rápida. Neste contexto, surge a Dívida Técnica (DT) [5], que referese a artefatos imaturos que foram produzidos durante o processo de desenvolvimento. A existência de DT em projeto de software é inevitável [19] e pode acontecer por diferentes motivos: *(i)* não admitida, quando por falta de experiência os desenvolvedores escrevem código de baixa qualidade; *(ii)* admitida, acontece quando o próprio desenvolvedor de forma deliberadamente causa a dívida, com o objetivo de priorizar atividades que necessitam de maior atenção. Neste trabalho, o foco é no tipo admitido, uma vez que este deixa sinais nos artefatos do software que podem ser identificados e atitudes preventivas podem ser tomadas. Uma delas é a identificação contínua de itens de DT e pagamento deles.

A presença de DT não está ligada diretamente apenas ao

artefato de código-fonte, outros artefatos que foram concebidos durante o desenvolvimento podem possuir débitos, alguns exemplos: documentação incompleta/insuficiente; baixa cobertura de casos de teste; quantidade de comentários insuficiente etc [2]. Uma das formas de identificação de DT auto-admitidas estudadas atualmente é a análise de comentários deixados em código-fonte [14, 8]. Comentários deixados no código-fonte são grande fonte de informação, porque os comentários podem descrever a lógica de determinado trecho da aplicação, também podem indicar possíveis melhorias a serem feitas ou relatar trechos do código fora de padrões e que não estejam funcionando de acordo com os requisitos planejados. A utilização de comentários para identificação de DT já se mostrou bastante útil, como no trabalho apresentado por Maldonado e Shihab [14] quantificaram e identificaram cinco diferentes tipos de DT auto-admitida manualmente em comentários de código-fonte: dívida de design; dívida de defeito; dívida de documentação; dívida de requisitos e dívida de teste. Farias *et al.* [8] propuseram uma ferramenta de mineração de DT em comentários – chamada eXcomment– e criaram um vocabulário contextualizado e heurísticas para tal fim, que foram incorporados na mesma. Este vocabulário foi introduzido na ferramenta de mineração de código eXcomment, a ferramenta utiliza técnica de processamento de linguagem natural para identificação de itens de DT. Embora uma prova de conceito tenha sido realizada para verificar como o vocabulário se comportava, os resultados indicaram que o vocabulário melhora na identificação de DT. Embora os estudos citados utilizem a mineração de comentários para identificação de itens de DT, ainda não há evidências nenhum estudo empírico avaliou o efeito do uso das heurísticas na identificação de DT pela ferramenta.

Neste contexto, este artigo explora as diferenças de duas versões da ferramenta eXcomment (com e sem heurísticas de vocabulário contextualizado) para avaliar o efeito das heurísticas objetivo principal da mesma. Além de ambas versões da ferramenta, utilizou-se para fins desta avaliação, o *dataset* de itens de DT em software de código-aberto disponibilizado por Maldonado e Shihab [14]. A avaliação do efeito das heurísticas foi realizado com relação às métricas de *precisão*, *cobertura* e *acurácia*.

Os resultados indicam que a segunda abordagem de fato reduziu a quantidade de falsos positivos e consequentemente melhorou *precisão*, *cobertura* e a *acurácia* da ferramenta. Entretanto, os resultados destacam a necessidade de melhorias na ferramenta para uso em ambiente de produção. As principais contribuições deste trabalho são: *(i)* a demonstração prática do potencial do uso das heurísticas; e *(ii)* a identificação de pontos de melhorias no dataset utilizado e na ferramenta.

O restante do trabalho está organizado como segue. Inicialmente, apresenta-se o referencial teórico (Seção 2) e em seguida o estudo de caso na Seção 3. A Seção **??** apresenta os resultados. A Seção 5 apresenta a discussão dos resultados. Enquanto as limitações e ameaças à validade do estudo são apresentadas nas Seções 6 5.3. Os trabalhos relacionados são apresentados na Seção 7 e por fim, a Seção 8 conclui o trabalho e a Seção 9 aponta trablhos futuros.

2. REFERENCIAL TEÓRICO

Esta seção apresenta os conceitos centrais deste trabalho, de maneira a facilitar o entendimento do seu conteúdo. Inicia-se com o conceito de dívida técnica e em seguida, apresenta-se a mineração de repositórios.

2.1 Dívida Técnica

O conceito de Dívida Técnica [5] contextualiza problemas nos artefatos que foram produzidos durante o processo de desenvolvimento, levando em conta atividades que não foram executadas de maneira correta e que consequentemente afetam a qualidade do software.

A presença de instâncias de dívida técnica pode acontecer por diferentes motivos: *(i)* não admitida, quando por falta de experiência os desenvolvedores escrevem código de baixa qualidade; *(ii)* admitida, acontece quando o próprio desenvolvedor de forma deliberadamente causa a dívida, com o objetivo de priorizar atividades que necessitam de maior atenção.

A ocorrência de dívida técnica pode acontecer por diferentes razões, sendos elas, *(i)* não admitida: quando a equipe de desenvolvimento não possui conhecimento suficiente para fazer realizar tal atividade, e de modo a escolher a solução que muitas das vezes não é adequada; *(ii)* intencional: Quando o gerente do time de desenvolvimento opta por fazer uma dívida para ganhar tempo ou para priorizar atividades que sejam mais importantes no momento. [8]

No estudo conduzido por Alves e seus colegas [2], é mostrado que as instâncias de dívida técnica não ocorrem apenas no código-fonte, ela também pode estar presente em outros artefatos que foram produzidos durante o desenvolvimento como: documentação, casos de teste e arquitetura mal planejada.

Alves *et al.* [3] realizou um estudo onde foram classificados diferentes tipos de dívida técnica ao todo foram 13 tipos. Na tabela x é apresentado os tipos de DT que serão considerados neste estudo.

2.2 Mineração de Repositórios

A mineração de repositórios de software (MSR) se refere ao estudo exploratório dos artefatos que foram produzidos durante o processo de desenvolvimento [13]. Repositórios de softwares são uma grande fonte de informações, pois temos acesso a todas versões do projeto, incluindo informações sobre todas as mudanças que ocorreram ao longo do da vida do repositório.

As principais etapas a serem seguidas no processo de MSR são: *(i)* Extração; *(ii)* Modelagem de Dados; *(iii)* Síntese; *(iv)* Análise [11]. Na primeira etapa é realizada a extração dos dados em sua forma mais bruta de diferentes tipos de repositórios. Na etapa seguinte que é a modelagem é realizada a preparação dos dados que serão utilizados durante o processo de mineração. Na etapa de síntese é realizado o processamento dos dados extraídos. E por fim, na etapa de análise vão ser interpretados os resultados e as possíveis conclusões.

Existem diferentes tipos de repositórios de software: controle de versão; bugs; log de implantação e código-fonte [13]. Somente o de código-fonte é de interesse deste estudo, pois temos acesso ao código-fonte e aos comentários. A escolha do repositório do tipo código-fonte permite a utilização de duas abordagens que são comumentes utilizadas para investigar itens de DT, sendo elas, (*i*) abordagem manual onde um grupo de pesquisadores irão fazer a leitura manual dos comentários com o objetivo de identificar itens de DT; (*ii*) abordagem automatizada, se deve através da utilização de ferramentas que possuem métricas pré-estabelecidas que fa-

zem análise do comentário com o objetivo de identificar DT.

3. ESTUDO DE CASO

Esta seção apresenta detalhes sobre as questões de pesquisa, o planejamento e a execução do estudo de caso conduzido.

3.1 Objetivo-Questão de pesquisa-Métricas

O objetivo deste estudo é *avaliar o efeito do uso de heurísticas na mineração de itens de dívida técnica com a* eXcomment, na perspectiva *do desenvolvedor de software*, com respeito à *quatro métricas: precisão, cobertura, f_1-score e acurácia*. Assim, definiu-se a seguinte questão de pesquisa: **Qual a efeito do uso das heurísticas na mineração de comentários de código-fonte realizada pela eXcomment?**

Para quantificar o efeito, a *precisão* vai indicar a razão entre os *verdadeiros positivos* (VP) pela soma deles à quantidade *falsos positivos* (FP) de *verdadeiros positivos*. Dentro do escopo do projeto seria a quantidade de DT que foi identificada corretamente sobre todas as DT que são consideradas verdadeiras. A equação 1 mostra o cálculo desta métrica.

$$precisão = \frac{VP}{VP + FP} \tag{1}$$

A *cobertura* representar o quanto as ferramentas acertaram na identificação de DT. Ou seja, a fração entre VP e soma dele ao erros de classificação da DT (FN). A equação 2 mostra o cálculo desta métrica.

$$cobertura = \frac{VP}{VP + FN} \tag{2}$$

A f_1-*score* representa uma média harmônica entre a *precisão* e a *cobertura*, que é calculada de acordo com a equação 3.

$$f_1 - score = \frac{2 * precisão * cobertura}{precisão + cobertura} \tag{3}$$

Por fim, a *acurácia* é métrica que pode ser representada pela, razão de previsões corretas sobre todas as previsões que foram feitas.

$$acurácia = \frac{VP + FP}{VP + VF + FN + FP} \tag{4}$$

Para cada umas dessas métricas, pode-se refinar a questão de pesquisa principal do estudo da seguinte forma:

RQ_p: Qual a efeito na *precisão* do uso das heurísticas na mineração de comentários de código-fonte realizada pela eXcomment?

RQ_c: Qual a efeito na *cobertura* do uso das heurísticas na mineração de comentários de código-fonte realizada pela eXcomment?

RQ_f: Qual a efeito na f_1-*score* do uso das heurísticas na mineração de comentários de código-fonte realizada pela eXcomment?

RQ_a: Qual a efeito na *acurácia* do uso das heurísticas na mineração de comentários de código-fonte realizada pela eXcomment?

3.2 eXComment: as diferenças entre as versões avaliadas

Como antecipado, utilizou-se uma versão da eXcomment com uma identificação ingênua (*sem a utilização das heurísticas – v_1*) e outra com identificação aprimorada (*com a utilização das heurísticas – v_2*). Embora a versão ingênua da ferramenta implemente algoritmos complexos de processamento de linguagem natural, optou-se pelo termo "ingênuo" para efeitos de diferenciação da estratégia utilizandose das heurísticas. As heurísticas da segunda versão contam com um vocabulário contextualizado chamado de CVM-TD (*Context Vocabulary Model on Technical Debt*), neste vocabulário possui termos e expressões que foram retirados dos comentários de código-fonte, que podem estar associados à dívida técnica, de projetos de código aberto [8].

A citação abaixo descreve detalhes sobre o desenvolvimento da versão ingênua.

> “ Uma primeira versão da eXcomment foi desenvolvida a fim de investigar a viabilidade de identificação de DT através de análise de comentários. Nessa versão, o vocabulário contextualizado foi usado para apoiar o processo de detecção de comentários que podem reportar uma situação de DT. Contudo, a ferramenta foi desenvolvida com algumas limitações, entre elas: *(i)* os padrões não eram detectados de maneira totalmente automática a fim de identificar os comentários que podiam indicar itens de DT. Os padrões eram inseridos de maneira manual ou semiautomática; *(ii)* não era possível calcular o total de *scores* de um comentário; e *(iii)* não era possível classificar os tipos de DT automaticamente. ”
>
> *Farias et al. [8]*

A citação abaixo descreve detalhes sobre o desenvolvimento da versão aprimorada.

> “ ... foram desenvolvidas novas funcionalidades, tornando possível analisar os comentários de um projeto de forma automatizada através de técnicas de mineração de texto. Para isso, foi criada uma integração entre a eXcomment e o CVM-TD, gerando a relação entre os padrões do vocabulário e os comentários onde os padrões são encontrados. Foram criadas heurísticas de busca com o objetivo de expandir o vocabulário. Por fim, criou-se uma forma de classificar os comentários usando os *scores* dos padrões e heurísticas encontrados neles, que possibilita ordenar os comentários de acordo com a possibilidade de indicar uma situação de DT. ”
>
> *Farias et al. [8]*

3.3 Sistemas-Alvo e Óráculo

Utilizou-se o oráculo construído por Maldonado e Shihab [14] de dois sistemas de código aberto. A Tabela 1 apresenta uma breve caracterização dos sistemas-alvo selecionados: JFreeChart e Apache Ant. Estes sistemas foram escolhidos por serem de código aberto, escritos em `Java`, pertencerem a domínio diferentes (Construção de Gráficos e Automação e compilação) e serem comumente utilizados em

estudos empíricos em engenharia de software. A Tabela descreve: *(i)* a versão utilizada nas análises; *(ii)* a quantidade de linhas de código de cada um dos sistemas; *(iii)* os comentários válidos considerados em cada uma das versões da eXcomment (*i.e.*, com e sem heurísticas); bem como *(iv)* a quantidade de itens de dívida técnica apontados no oráculo.

Maldonado e Shihab [14] nomearam *Dívida de Implementação* o que é considerado pela eXcomment *Dívida de Requisitos*. Deste ponto em diante utilizar-se-á somente *Dívida de Requisitos*. O tipo *Dívida de Requisitos* (DT1) refere-se a atividades que o time de desenvolvimento tem para implementar e como será realizada essa implementação. A *Dívida de Defeito* (DT2) está relacionada à defeitos conhecidos, independentemente se reportada por usuários ou testes, dos quais o time concorda que deveriam ser corrigidos. O tipo *Dívida de Design* (DT3) se refere às más práticas de codificação que violam os princípios da orientação a objetos. Por fim a *Dívida de Teste* (DT4), está relacionada a problemas que podem afetar a qualidade das atividades de teste [3].

3.4 Coleta de dados

Os dois primeiros autores do artigo fizeram a inspeção manual dos itens de DT presentes no oráculo comparando com os itens identificados automaticamente pela eXcomment. Esta inspeção manual foi conduzida da seguinte maneira: *(i)* o inspetor selecionava o comentário que a ferramenta marcou com uma DT; *(ii)* o inspetor utilizava o comentário selecionado para identificar o mesmo no oráculo. Este processo foi repetido para todos os comentários dos sistemas alvos.

4. RESULTADOS

Esta seção apresenta os resultados do estudo experimental. Para melhor apresentação, os dados de cada um dos sistemas-alvo são apresentados em uma seção separada (*i.e.*, primeiro o JFreeChart e em seguida o Apache Ant). Além disso, apresenta-se resultado em duas formas: dados crus (Tabelas 2 e 4) e na forma das métricas utilizadas nas questões de pesquisa (Tabelas 3 e 5). Todas estas quatro tabelas possuem (sub-)colunas indicando v_1 e v_2 para efeito de comparação de resultados entre as duas versões avaliadas.

4.1 JFreeChart

A Tabela 2 apresenta os resultados crus, enquanto a Tabela 3 apresenta as métricas de interesse calculadas com relação ao JFreeChart. Observando as duas versões na coluna VP, percebe-se que a abordagem que faz o uso de heurísticas teve um desempenho melhor identificação de DT. Outro ponto a se levar em conta é a quantidade de itens marcados como FP, novamente, a abordagem com heurística teve um desempenho melhor. Enquanto a abordagem com heurística marcou 291 itens como FP e abordagem sem heurística marcou 1107, uma diferença de 816 itens a favor da v_2. Com relação aos VN, ambas versões tiveram uma diferença considerável, possuindo uma diferença de 881 itens, no agregado, mas mínima se considerados cada tipo de dívida individualmente. Por fim, a última coluna mostra a quantidade de itens de FN, percebe-se que as duas versões tiveram resultados similares, possuindo apenas um item de diferença no resultado agregado.

Em termos gerais, a eXcomment conseguiu ser mais efetiva ao identificar itens de *Dívida de Defeito*. A abordagem que utiliza heurística conseguiu identificar corretamente um único item de *Dívida de Requisitos* e de *Dívida de Design*. A grande quantidade de FP afeta diretamente o resultado das métricas de *precisão* e f_1-*score* (Tabela 3).

4.2 Apache Ant

A Tabela 4 apresenta os resultados crus, enquanto a Tabela 5 apresenta as métricas de interesse calculadas com relação ao Apache Ant. Analisando a coluna VP, percebe-se que a abordagem que utiliza heurísticas teve um desempenho melhor em relação a outra abordagem, marcando quatro itens de DT a mais que a outra versão utilizada na comparação. O tipo de DT que teve a maior quantidade de itens corretamente identificados foi novamente o de *Dívida de Defeito*.

Os dados presentes na coluna FP mostram que a abordagem que não faz o uso de heurística teve um desempenho inferior em relação a outra abordagem, clasificando incorretamente 1268 itens a mais. Sendo que o tipo DT que possui maior ocorrênciade FP é a de *Dívida de Defeito*, seguida pela *Dívida de Teste* e pela *Dívida de Design*. A quantidade de ocorrências de FP relacionadas à *Dívida de Teste* teve uma redução considerável na abordagem que faz o uso de heurística, reduzindo de 593 itens de para apenas dois.

A coluna de resultados relacionados aos VN, mostra que a segunda abordagem marcou mais itens em relação à primeira abordagem, apresentando também uma diferença considerável de 1563 itens. Por fim, a única coluna em que os dados das duas versões são praticamente iguais é na coluna de FN, apresentando apenas uma diferença de três itens erroneamente não classificados como itens de *Dívida de Requisitos*, *Dívida de Defeito* e *Dívida de Teste*.

5. DISCUSSÃO

5.1 Respostas à questão de pesquisa

Pretende-se responder a questão de pesquisa de forma indireta, respondendo cada uma das sub-questões. Para isto, utilizaram-se os dados apresentados nas Tabelas **??** e **??**.

Com relação à *precisão*, vê-se que ambas abordagens de mineração de itens de DT foram *imprecisas* e não houve diferença significativa entre as abordagens. Os autores acreditam que existem duas principais causas para o problema da grande quantidade de *falsos positivos*, que impacta diretamente no resultado de *precisão*: tanto o viés dos pesquisadores que criaram o oráculo (incluindo o fato de apontarem apenas um tipo de DT por comentário), quanto a possível existência de um vocabulário próprio para cada projeto de software.

Com relação à *cobertura*, vê-se que ambas abordagens de mineração de itens de DT tiveram *pouca cobertura*, no entanto percebeu-se diferenças significativas para a identificação de itens de *Dívida de Defeito*, onde a *cobertura* no JFreeChart triplicou e no Apache Ant duplicou. Este resultado indica que a ferramenta ainda carece de ajustes quanto a identificação destes tipos de DT em ambientes de produção.

Com relação à f_1-*score*, é sabido que este depende diretamente dos resultados de *precisão* e *cobertura*. Como ambas as abordagens à mineração de itens de DT foram imprecisas e tiveram pouca cobertura nos cenários avaliados, já era esperado que a média entre eles fosse também baixa.

Com relação à *acurácia*, vê-se que ambas abordagens de mineração de itens de DT tiveram bons resultados, podendo

Table 1: Caracterização dos sistemas-alvo.

Sistema	Versão	Método	LOC	Comentários	DT1	DT2	DT3	DT4
JFreeChart	1.5.0	v_1	144342	15999	25	9	183	1
		v_2	144342	15608				
Apache Ant	1.0.19	v_1	115851	14512	16	13	95	10
		v_2	115851	13812				

LOC: Tamanho em linhas de código; v_1**:** Sem heurística; v_2 Com heurística; **DT1**: *Dívida de Requisitos*; **DT2**: *Dívida de Defeito*; **DT3**: *Dívida de Design*; **DT4**: *Dívida de Teste*.

Table 2: Dados crus do JFreeChart.

Tipos de DT	VP		FP		VN		FN	
	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2
Dívida de Requisitos	0	1	5	33	11899	11520	24	24
Dívida de Defeito	1	3	145	149	11774	11420	8	6
Dívida de Design	1	1	225	176	11520	11219	182	182
Dívida de Teste	1	0	883	0	11094	11577	0	1
`Total`	3	5	1258	358	46287	45736	214	213

DT: Dívida Técnica **VP:** Verdeiro positivo; **FP:** Falso positivo; **VN:** Verdadeiro negativo; **FN:** Falso negativo.

ser observado alguma melhoria nos resultados, especialmente no caso do Apache Ant.

5.2 Análise dos Falsos Positivos

Nesta sessão se faz a discussão dos diferentes tipos de *falsos positivos* (*FP*) que foram encontrados na análise dos resultados dos estudos de caso. Com a análise dos *FP*, conseguiu-se definir três principais classes: A Tabela 6 identifica e define cada uma destas classes. *FP1*– quando a eXcomment marcou como DT, mas o oráculo possuía não indicava quaisquer classifação; *FP2*– quando a eXcomment marcou como tipo A, mas no oráculo tinha tipo B de DT; e *FP3*– quando a eXcomment marcou como DT, mas o comentário não existia no oráculo. Resolveu-se investigar os *scores* produzidos pela v_2 da eXcomment.

As Figuras 1a e 1b e a Tabela 7 apresentam distribuição dos *FP* de acordo com o valor dos *scores* de cada item de DT. Os *scores* dos diferentes tipos de *FP* que foram encontrados se assemelham à distribuição dos scores do *VP*. Esse fato corrobora com a expectativa apresentada por Farias *et al.* [8], na qual os comentários que possuem maior *score* são os comentários que possuem maior chance de indicar presença de DT. É possível observar inclusive que no caso do Apache Ant (Figura 1b), muitos dos comentários classificados como itens de dívida *FP1*, *FP2* e *FP3* são inclusive maiores que os valores daqueles identificados como *VP*. Estes padrões de *score* se repetem também se analisarmos cada um dos típos de DT em separado, omitiu-se os gráficos devido a limitação de espaço.

Este fato levanta a possibilidade de que o *oráculo* possa conter vieses de construção. De fato, ao revisitar o processo de construção do mesmo [14], percebeu-se que o processo de construção do oráculo se deu por um único pesquisador, que por si só não é uma prática muito aconselhada, pois a classificação dos itens de DT podem ser enviesada pelo ponto de vista do pesquisador. Junte-se a isto, o fato de apontarem apenas um tipo de DT por comentário, o que pode ter forçado à análise automatizada pela ferramenta a incorrer em muitos falsos positivos, uma vez que a mesma indica mais de um tipo de dívida por comentário. Vale ressaltar que, a classificação dos itens de dívida técnica pela eXcomment foi avaliada manualmente por pelo menos dois pesquisadores experientes [7].

Outro fato a ser levantado é que a eXcomment consegue realizar a identificação de novetipos de DT sendo eles: *Dívida da Arquitetura*; *Dívida de Construção*; *Dívida de Código*; *Dívida de Defeito*; *Dívida de Design*; *Dívida de Documentação*; *Dívida de Pessoas*; *Dívida de Requisitos*; *Dívida de Teste*. Enquanto no oráculo, apenas cinco tipos foram considerados: *Dívida de Design*; *Dívida de Defeito*; *Dívida de Documentação*; *Dívida de Requisitos*. Dos cinco tipos presente no oráculo, apenas quatro tipos estavam presentes nos sistemas alvos deste estudo. Alguns comentários foram marcados como possíveis indicadores de DT, porém o tipo da dívida que a eXcomment marcou divergia com o do oráculo, esse tipo *FP* foi classificado como *FP2*. O tipo de *FP2* merece uma investigação especial, devido ao fato da discordância entre a eXcomment e o oráculo, seria necessário a leitura manual dos comentários para ver qual abordagem está mais próxima do resultado correto. Ao observar as Figuras 1b e 1a é possível perceber que uma parcela considerável de comentários que podem indicar DT foram considerados como o tipo *FP2*, o que pode ter contribuído para o número elevado de itens de erroneamente classificados.

Além disso, recentemente em pesquisa independente desta, Farias [9] investigaram padrões que produzem falsos positivos na eXcomment. Os padrões encontrados mostram que a ferramenta ainda não consegue entender por completo o contexto de onde o comentário está inserido. Os autores também apresentam o fato de que cada repositório possui um vocabulário próprio, isso pode indicar que se um repositório possuir um conjunto de palavras próprio, caso essas

Table 3: Métricas do JFreeChart.

Dívida Técnica	*precisão*		*cobertura*		*f_1-score*		*acurácia*	
	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2
Dívida de Requisitos	0.0%	3.0%	0.0%	4.0%	0.0%	3.0%	100.0%	100.0%
Dívida de Defeito	1.0%	2.0%	11.0%	33.0%	1.0%	4.0%	99.0%	99.0%
Dívida de Design	0.0%	1.0%	1.0%	1.0%	0.0%	1.0%	97.0%	97.0%
Dívida de Teste	0.0%	0.0%	100.0%	0.0%	0.0%	0.0%	93.0%	100.0%

Table 4: Dados crus do Apache Ant.

Dívida Técnica	VP		FP		VN		FN	
	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2
Dívida de Requisitos	0	0	0	29	13282	12625	16	16
Dívida de Defeito	4	8	899	477	12391	12180	9	5
Dívida de Design	1	4	255	74	12953	12501	94	91
Dívida de Teste	3	0	593	2	12700	12658	7	10
`Total`	8	12	1747	479	50966	49964	126	122

VP: Verdadeiro positivo; **FP:** Falso positivo; **VN:** Verdadeiro negativo; **FN:** Falso negativo.

palavras estejam presentes no vocabulário a ferramenta irá considerá como um falso positivo, fazendo com que vários comentários que não possuem presença de DT sejam selecionados.

Dado os fatos mencionados anteriormente, percebe-se que vários fatores influenciaram na quantidade de falsos positivos, em resumo os principais fatores são: *(i)* oráculo construído por apenas um pesquisador; *(ii)* a eXcomment pode indicar para o mesmo comentários vários tipos de DT; *(iii)* vocabulário próprio do repositório. A quantidade *FP* afeta as métricas utilizadas nas questões de pesquisa, levando em conta que os tipos que foram identificados como *FP2* não foram considerados no *oráculo*, a quantidade *FP* poderia ter sido menor e merece investigações adicionais.

5.3 Limitações

Os autores estão cientes de que o número de repositórios avaliados, bem como a quantidade de tipos de DT avaliadas em relação ao que a ferramenta é capaz de identificar (somente quatro das nove) limita o poder de conclusões sobre o efeito das heurísticas na identificação de itens de DT. No entanto, o estudo possibilitou identificar também a limitação dos repositórios de DT disponíveis na literatura com relação a serem utilizados como padrão ouro em estudos empíricos. Construir um *dataset* robusto e sem vieses é um desafio não-trivial.

6. AMEAÇAS À VALIDADE

Constantemente, estudos empíricos são conduzidos sob condições não ideais devido à inúmeras restrições identificadas durante o processo de investigação. Portanto, se faz necessário a discussão de algumas ameaças à validade deste estudo, bem como as atitudes tomadas para contornar ou minimizar os efeitos destas.

Validade de conclusão. Infelizmente, a quantidade de tipos de dívidas disponíveis no oráculo, bem como o número de sistemas-alvo avaliados, não dá possibilidade de apresentarmos conclusões com significância estatística sobre o efeito das heurísticas na identificação de dívida técnica. No entanto, os resultados podem ser vistos como ponto de referência para pesquisas de replicação deste estudo em busca desta significância.

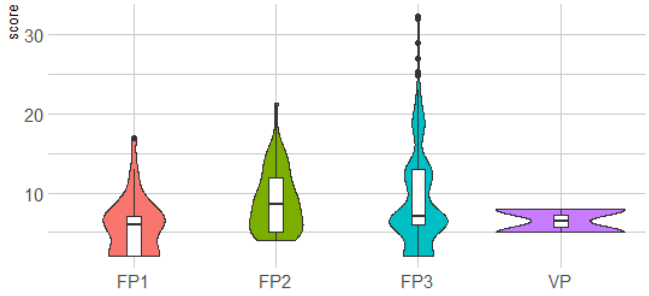
Validade interna. *(i)* viés na elaboração do *dataset* de dívidas técnicas: infelizmente, o *dataset* foi construído por apenas uma pessoa, embora com experiência comprovada, a classificação dos itens de dívida técnica podem tender a conter viéses da experiência do elaborador. No entanto, como o manuseio dos artefatos do estudo foi feita por alunos de graduação, que possuem pouca experiência, optou-se por utilizar um dataset já revisado e aceito pela comunidade para minizar o impacto desta ameaça na construção do oráculo. *(ii)* incompletude do dataset: os autores do *dataset* somente associaram um único tipo de dívida técnica por comentário, o que pode ter forçado à análise automatizada pela ferramenta a incorrer em muitos falsos positivos, uma vez que a mesma indica mais de um tipo de dívida por comentário. Para minimizar esta ameaça, apresentamos uma discussão sobre a alta ocorrência de falsos positivos identificados.

Validade externa. os autores asseguram que a única alteração entre as duas versões das ferramentas executadas fosse a adição das heurísticas de vocabulários específicos para a identificação dos itens de dívida técnica. Assim, estamos seguros que é possível de que existe uma relação de causa-efeito nas diferenças entre os resultados observados.

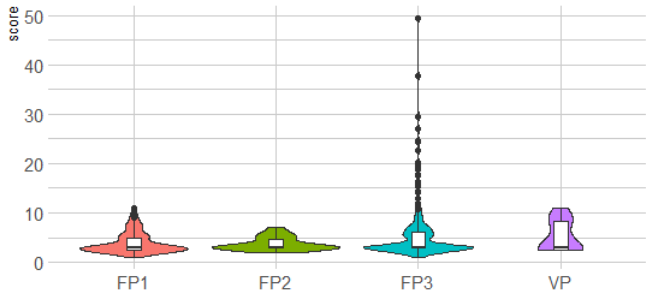
7. TRABALHOS RELACIONADOS

Table 5: Métricas do Apache Ant.

Dívida Técnica	*precisão*		*cobertura*		f_1-*score*		*acurácia*	
	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2
Dívida de Requisitos	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	99.8%	99.6%
Dívida de Defeito	0.4%	1.6%	30.8%	61.5%	0.9%	3.2%	93.2%	96.2%
Dívida de Design	0.4%	5.1%	1.1%	4.2%	0.6%	4.6%	97.4%	98.7%
Dívida de Teste	0.5%	0.0 %	30.0%	0.0%	1.0%	0.0%	95.5%	99.9%



(a) *Violin chart* JFreeChart com heurística.



(b) *Violin chart* Apache Ant com heurística.

Figure 1: Distribuição de Falsos Positivos.

Table 6: Tipos de falsos positivos.

Tipo	Descrição
FP1	eXcomment marcou como DT, mas o oráculo possuia *WITHOUT*, que indica sem classifação.
FP2	eXcomment marcou como tipo A, mas no oráculo tinha tipo B.
FP3	eXcomment marcou como DT, mas não existia no oráculo.

***FP*:** Falso Positivo

Dívida técnica é um tópico recorrente nos fóruns e publicações periódicas de engenharia de software [1]. Embora muitos desafios ainda resistam, este trabalho, especificamente, tem objetivo de avaliar o uso do efeito de heurísticas na mineração de DT em comentários de código-fonte. Durante nossa revisão não foram identificados trabalhos que investigam o mesmo problema. Assim, serão apresentados a seguir trabalhos que são considerados relacionados a este mesmo que atacando desafios distintos [19, 14, 7, 17, 6, 15, 10, 16].

Diversos trabalhos têm estudado a identificação e gerenciamento de DT [19, 4, 12, 18, 1]. Esses trabalhos fornecem indicadores de como itens de DT podem aparecer em um projeto e quais medidas devem ser tomadas para que a situação não saia do controle. Com o objetivo de fornecer uma organização para os tipos de DT, Alves *et al.* [3] propôs uma ontologia sobre os diferentes tipos DT. O estudo reúne as definições sobre os tipos DT que são comumente utilizados e seus possíveis indicadores.

Passos *et al.* [17] identificaram DT através da mineração de comentários de código-fonte. Os resultados do estudo indicam que é possível identificar DT no documento de requisitos através da mineração de comentários. Em outro trabalho, Farias *et al.* [8] apresentam o *Contextualized Vocabulary Model on TD* (CVM-TD), este vocabulário possui termos que os desenvolvedores usam que podem indicar presença de DT auto-admitida, esse vocabulário foi adicionado à eXcomment. Foi realizado um teste de conceito utilizando dois projetos de software open source, ArgoUml e

Table 7: Detalhamento da quantidade de *falsos positivos* por classe em cada um dos sistemas-alvo.

Dívida Técnica	JFreeChart						Apache Ant					
	FP1		*FP2*		*FP3*		*FP1*		*FP2*		*FP3*	
	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2	v_1	v_2
Dívida de Requisitos	1	2	0	1	4	30	2	10	0	0	3	19
Dívida de Defeito	8	6	9	24	128	109	76	67	13	11	813	399
Dívida de Design	24	23	2	1	199	152	31	14	4	0	220	60
Dívida de Teste	55	0	2	0	776	0	75	1	7	0	511	1
Total	88	31	13	26	1107	291	184	92	24	11	1547	479

v_1: Sem Heurísticas; **v_2:** Com Heurísticas;

JFreeChart. Os resultados indicam que os comentários minerados realmente podem indicar presença de DT auto-admitida.

Xavier *et al.* [20] mostraram que somente 29% dos problemas reportados em sistemas de controle *bugs* marcados como dívida técnica podem ser rastreadas para os comentários de código. Eles mostraram ainda que este tipo de dívida leva mais tempo para serem atendidos, mesmo que não sejam mais complexos em termos de *code churn*. Em outro trabalho, Oliveira *et al.* [16] compararam ferramentas de identificação de dívida técnica, com destaque para abordagem baseada em análise estática *(code smells)* e comentários de código-fonte feito por desenvolvedores. Para a realização do estudo foram utilizados 1.000 repositórios na linguagem Python, utilizando-se do SATDDetector um plug-in que faz uso de aprendizado de máquina para a coleta de comentários de código-fonte que possivelmente contenham dívida, foi utilizada também o SonarQube, ferramenta que faz a identificação automática de dívida técnica, a fim de traçar um paralelo entre ambas as versões. Como resultado, foi observado que 19,47% das dívidas técnicas que foram analisadas são provenientes de comentários admitidos por desenvolvedores que também são identificadas pelo SonarQube, estabelecendo possivelmente uma relação complementar entre ambas as versões.

Área de pesquisa de DT é relativamente nova por esse motivo alguns trabalhos vêm tentando classificar os vários tipos de DT, que faz com que a classificação dos itens de DT fiquem dispersos na literatura, visando fornecer uma organização para os tipos Alves *et al.* [3] propôs uma antologia que tinha como o objetivo organizar as diferentes classificações que estavam espalhadas literatura, o estudo em questão fornece a definição e os possíveis indicadores para que essas DT ocorram, ao todo foram classifcados 13 tipos de DT, por exemplo: DT de documentação, DT de teste, DT de código, DT de requisitos, etc. As definições fornecidas pelo trabalho anterior serviram como base para outros trabalhos como, o estudo proposto por maldonado a shihab [14], que tinham como o objetivo realizar a identificação de DT auto-admitida em repositórios de código-fonte, para realização o estudo foram analisado os seguintes repositórios Apache Ant, Apache Jmeter, ArgoUml, columba JFreeChart. Um dos critérios que foi adotado para seleção dos repositórios é que os mesmos deveriam pertecer a diferentes donimios e tamanhos variados, ao todo foram analisados 166 mil comentários, foram identificados 5 tipos de DT auto-admitida sendo elas: DT de documentação, DT de teste, DT de requisitos, DT de defeito e DT de design.

Um das principais maneiras de entender como um área se encontra é se utilizando de estudo terciários, onde os autores irão apresentar o estado atual da arte, como o trabalho elaborado por rios seus colegas [1] investigaram o estado atual da arte no campo da DT. O estudo promoveu contribuições, sendo elas, ter evoluído os tipos conhecidos de DT, fornecendo uma definição mais formal para cada item. Outra contribuição é o direcionamento para campos que ainda necessitam de uma investigação. Por fim, os tópicos para os quais os pesquisadores têm uma atenção maior são identificação e definição de DT.

Existem diferentes maneiras para realizar a identificação DT em um projeto de software, a comumente mais utilizada é a forma automatizada onde uma ferramenta vai analisar o repositório utilizando métricas pré-estabelecidas para identificação de DT, como é apresentado nos trabalhos [documentação, visminer TD]. Se utilizando da abordagem automatizada Passos e seus colegas [17] tinham como objetivo identificar DT no documento de requisitos através da mineração de comentários de código-fonte, para realização do estudo utilizou-se a ferramenta eXcomment, e os repositórios selecionados foram o argoUML e o JfreeChart, os resultados obtidos indicam que é possível identificar DT no documento requisitos através dos comentários do código-fonte, outra contribuição que o autor cita é a descoberta de novos padrões que podem ser adicionados na eXcomment, fazendo que com isso tenha uma maior cobertura dos itens de DT. O estudo elaborado por Mendes e seus colegas [15] apresentou a ferramenta VisminerTD uma ferramenta que permite mineração, identificação e monitoramento de itens de DT, a ferramenta se utiliza de submódulos para fazer a identificação de DT como: eXcomment, repositoryminer. Para validar a ideia da ferramenta foi feito um estudo de caso, onde foram minerados alguns repositórios. Os resultados obtidos indicam que a VisminerTD é uma boa opção que pode auxiliar na identificação e monitoramento de itens de DT.

F. Farias *et al.* [8] e seus colegas apresentam os conceitos de dívida técnica (DT), bem como a importância de seu gerenciamento e controle para a manutenção do software. É discutido como dívida técnica pode ser identificada por meio de comentários de desenvolvedores no código-fonte. Nesse contexto é apresentado um vocabulário contextualizado para a detecção de dívida técnica denominado Context Vocabulary Model on Technical Debt (CVM-TD). Apresentaram também a ferramenta de mineração de código-fonte eXcom-

ment integrada ao vocabulário contextualizado CVM-TD. Em seguida é proposto um teste de conceito utilizando dois projetos de software open source, a saber, ArgoUML e JFreeChart. Os resultados obtidos são importantes para a identificação de DT em projetos de software e a definir como e por onde começar a pagar as dívidas.

Felipe Gustavo de S. Gomes *et al.* [6] e seus colegas investigam a relação entre code smells e dívida técnica auto admitida (DTAA). Para a condução do estudo foram utilizados três projetos open source desenvolvidos em Java, a saber, ArgoUML, JFreeChart e Apache Ant. Os projetos foram escolhidos devido a sua ampla utilização em estudos científicos acerca de comentários de código-fonte. [6] Felipe Gustavo de S. Gomes e seus colegas descobriram forte tendência em relação a dívida técnica auto admitida e code smells e que em alguns casos a utilização de comentários de código-fonte pode complementar informações que não poderiam ser obtidas apenas com o uso de code smells.

8. CONCLUSÃO

Neste estudo investigou-se o efeito do uso de heurísticas na mineração de DT em comentários de código-fonte. O estudo apresentou evidências de como o uso de um vocabulário contextualizado melhora a identificação de itens de DT em comentários de código-fonte. O uso de heurísticas obteve sucesso na redução do número de *falsos positivos* da ferramenta eXcomment, mas aparentemente ainda não foi o suficiente para permitir o seu uso na identificação de itens de DT em ambiente de produção. Por outro lado, os resultados instigam a investigação do problema por outras perspectivas, tais como a construção de um oráculo mais robusto e por várias mãos para minimizar os casos de discordância entre os tipos de dívida identificados. Acredita-se que a eXcomment têm potencial para identificação de DT, mas carece de mais esforços para a eliminação/minimização da quantidade de *FP* produzidos.

9. TRABALHOS FUTUROS

Com trabalhos futuros, é possível ampliar a base de projetos, para se ter um aspecto mais amplo do uso de efeitos heurísticas, bem como utilizar de inteligência artificial para detectar padrões de código-fonte para delimitação dos contextos em que os comentários estão inseridos. Além disso, se faz necessário a construção de oráculos mais robustos.

10. REFERENCES

[1] Nicolli S. R. Alves, Manoel Gomes de Mendonça Neto, and Rodrigo Oliveira Spínola. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information Software Technologies*, 102:117–145, 2018.

[2] Nicolli SR Alves, Thiago S Mendes, Manoel G de Mendonça, Rodrigo O Spínola, Forrest Shull, and Carolyn Seaman. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70:100–121, 2016.

[3] Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes, and Rodrigo O Spínola. Towards an ontology of terms on technical debt. In *2014 Sixth International Workshop on Managing Technical Debt*, pages 1–7. IEEE, 2014.

[4] Gabriele Bavota and Barbara Russo. A large-scale empirical study on self-admitted technical debt. In *Proceedings of the 13th international conference on mining software repositories*, pages 315–326, 2016.

[5] Ward Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1992.

[6] Felipe Gustavo de S. Gomes, Thiago Souto Mendes, Rodrigo O. Spínola, Manoel Mendonça, and Mário Farias. Uma análise da relação entre code smells e dívida técnica auto-admitida. In *Anais do VII Workshop on Software Visualization, Evolution and Maintenance (VEM)*, pages 37–44, Porto Alegre, RS, Brasil, 2019. SBC.

[7] Mário André Freitas Farias, Manoel Gomes de Mendonça Neto, Marcos Kalinowski, and Rodrigo Oliveira Spínola. Identifying self-admitted technical debt through code comment analysis with a contextualized vocabulary. *Information Software Technologies*, 121:106270, 2020.

[8] Mário André Freitas Farias, Railan Xisto, Marcos S. Santos, Raphael Silva Fontes, Methanias Colaço Jr., Rodrigo O. Spínola, and Manoel G. Mendonça. Identifying technical debt through a code comment mining tool. In Fábio Gomes Rocha, Igor Vasconcelos, Rodrigo Pereira dos Santos, Davi Viana, and Scheila de Avila e Silva, editors, *Proceedings of the XV Brazilian Symposium on Information Systems, SBSI 2019, Aracaju, Brazil, May 20-24, 2019*, pages 18:1–18:8. ACM, 2019.

[9] Mário Farias, Thiago Souto Mendes, Monoel G. Mendonça, and Rodrigo O. Spínola. On comment patterns that are good indicators of the presence of self-admitted technical debt and those that lead to false positive items. In *Proceedings of the 27th Annual Americas Conference on Information Systems (AMCIS)*, pages 1–10, 2021.

[10] Sávio Freire, Nicolli Rios, Manoel Mendonça, Davide Falessi, Carolyn Seaman, Clemente Izurieta, and Rodrigo O Spínola. Actions and impediments for technical debt prevention: results from a global family of industrial surveys. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 1548–1555, 2020.

[11] Hadi Hemmati, Sarah Nadi, Olga Baysal, Oleksii Kononenko, Wei Wang, Reid Holmes, and Michael W Godfrey. The msr cookbook: Mining a decade of research. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 343–352. IEEE, 2013.

[12] Qiao Huang, Emad Shihab, Xin Xia, David Lo, and Shanping Li. Identifying self-admitted technical debt in open source projects using text mining. *Empirical Software Engineering*, 23(1):418–451, 2018.

[13] Huzefa Kagdi, Michael L Collard, and Jonathan I Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*, 19(2):77–131, 2007.

[14] E. D. S. Maldonado and E. Shihab. Detecting and quantifying different types of self-admitted technical debt. In *Proceedings of the 7th Workshop on Managing*

Technical Debt, MTD '11, pages 9–15, New York, NY, USA, 2015. IEEE.

[15] Thiago S. Mendes, Felipe G. S. Gomes, David P. Gonçalves, Manoel G. Mendonça, Renato L. Novais, and Rodrigo Oliveira Spínola. Visminertd: a tool for automatic identification and interactive monitoring of the evolution of technical debt items. *Journal of the Brazilian Computer Society*, 25(1):2, 2019.

[16] Isabela Oliveira, Humberto Marques-Neto, and Laerte Xavier. Analisando estratégias para identificação de dívidas técnicas. In *Anais do VIII Workshop de Visualização, Evolução e Manutenção de Software*, pages 9–16, Porto Alegre, RS, Brasil, 2020. SBC.

[17] Amanda F. O. Passos, Mário André Freitas Farias, Manoel G de Mendonça Neto, and Rodrigo Oliveira Spínola. A study on identification of documentation and requirement technical debt through code comment analysis. In *Proceedings of the 17th Brazilian Symposium on Software Quality*, pages 21–30, 2018.

[18] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. Neural network-based detection of self-admitted technical debt: From performance to explainability. *ACM transactions on software engineering and methodology (TOSEM)*, 28(3):1–45, 2019.

[19] Edith Tom, AybüKe Aurum, and Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516, 2013.

[20] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. Beyond the code: Mining self-admitted technical debt in issue tracker systems. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 137–146, 2020.