VisminerTD: Uma Ferramenta para Identificação Automática e Monitoramento Interativo de Dívida Técnica

Thiago S. Mendes[1,2], David P. Gonçalves[1], Felipe G. Gomes[1], Renato Novais[2]
Rodrigo O. Spínola[3], Manoel Mendonça[1]

[1]Universidade Federal da Bahia, [2]Instituto Federal da Bahia, [3]Universidade Salvador
Salvador, BA.

{thiagomendes, davidpinho, felipegustavo}@dcc.ufba.br, renato@ifba.edu.br

rodrigo.spinola@pro.unifacs.br, manoel.mendonca@ufba.br

Resumo. *Dívida técnica descreve as tarefas de desenvolvimento pendentes em projetos de software, que trazem benefícios a curto prazo, mas que podem causar prejuízos no futuro se não forem tratadas. Sua presença traz riscos para o projeto, podendo reduzir sua qualidade. Para gerenciá-la é importante ter mecanismos automáticos que suportem a análise de muitos dados. Desta forma, técnicas de visualização de software se mostram um caminho eficiente para lidar com as dívidas. Este artigo apresenta a ferramenta VisminerTD, que permite a identificação automática e monitoramento interativo da evolução de itens da dívida. Para avaliar sua aplicabilidade, foi realizado um estudo de viabilidade para analisar o código fonte do projeto JUnit. Os resultados indicaram que a ferramenta pode apoiar a equipe de desenvolvimento a lidar com os itens de dívida.*

Abstract. *Technical debt describes pending development tasks in software projects, which bring short-term benefits, but that can cause problems in the future if not treated. Its presence brings risks to the project and can reduce its quality. To manage it, it is important to have automatic mechanisms that support the analysis of many data. In this sense, software visualization techniques can be an efficient way to deal with debt items. This paper presents the VisminerTD tool, which allows automatic identification and interactive monitoring of the evolution of debt items. To evaluate its applicability, a feasibility study was carried out to analyze the source code of the JUnit project. The results indicated that the tool can support the development team in dealing with debt items.*

1. Introdução

Dívida Técnica (DT) representa o problema das tarefas de desenvolvimento pendentes como um tipo de dívida que traz um benefício a curto prazo para o projeto, normalmente em termos de maior produtividade e entregas mais rápidas de versões do produto, mas que pode ter de ser paga com juros mais tarde no processo de desenvolvimento [Seaman and Guo 2011][Izurieta et al. 2012]. O uso do conceito facilita a compreensão dos custos causados por decisões realizadas durante o ciclo de vida de desenvolvimento do software [Cunningham 1992].

Os efeitos da DT são revelados como problemas durante a evolução do software e podem afetar diferentes etapas do desenvolvimento devido aos diferentes tipos de dívida

existentes [Alves et al. 2016]. A presença de itens de dívida, situação enfrentada em diversos sistemas de informação em desenvolvimento, torna mais difícil a adição de novos requisitos, cria um ambiente favorável para o surgimento de defeitos, impacta na qualidade externa e reduz a manutenibilidade do código [Spínola et al. 2013]. Esse cenário tem estimulado a realização de esforços no sentido de desenvolver estratégias para identificar e gerir a DT [Alves et al. 2016].

O processo de gestão da DT é formado por três atividades principais: identificação, medição e monitoramento [Guo et al. 2014]. As atividades de identificação podem ser realizadas de forma manual ou automática [Zazworka et al. 2013]. O uso de técnicas de visualização de software pode ser feito como um apoio adicional para a tarefa de compreender, manter e evoluir sistemas de software [Novais et al. 2012][Magnavita et al. 2016]. Entretanto, essas técnicas ainda têm sido utilizadas de forma muito limitada para apoiar a identificação e monitoramento dos itens de dívida em projetos de software [Alves et al. 2016].

Neste contexto, este trabalho apresenta o desenvolvimento da ferramenta VisminerTD, cujo o objetivo é apoiar as atividades de identificação e monitoramento de DT utilizando recursos de visualização de software. O VisminerTD[1] implementa uma nova estratégia de identificação de DT por meio da combinação de informações de métricas de software e comentários de código fonte. Um estudo de viabilidade foi realizado considerando um projeto *open source* para analisar VisminerTD com o propósito de caracterizá-la em relação à viabilidade de uso no apoio às atividades de identificação e monitoramento de evolução dos itens de DT. Os resultados do estudo indicaram que a ferramenta pode apoiar engenheiros de software a lidarem com os itens de dívida.

Além desta introdução, este artigo possui mais quatro seções. A Seção 2 descreve a arquitetura e as funcionalidades do VisminerTD. A Seção 3 apresenta o estudo de viabilidade realizado. A Seção 4 discute alguns trabalhos relacionados. Por fim, a Seção 5 conclui o trabalho e destaca trabalhos futuros.

2. A Ferramenta VisminerTD

VisminerTD é uma ferramenta web *open source*, que possui múltiplas perspectivas visuais para apoiar a equipe de desenvolvimento nas atividades de identificação e monitoramento da evolução de itens de DT. Para isso, ela exibe informações sobre a estrutura e a evolução de projetos de software a partir de dados de métricas e comentários extraídos de repositórios de código fonte. A Figura 1 apresenta uma visão arquitetural componente & conector (estrutural) da ferramenta.

Todos os dados que são apresentados em VisminerTD são analisados e extraídos pelo módulo RepositoryMiner. Para a definição de quais informações seriam mineradas dos repositórios de software para permitir a identificação de itens de DT, foi considerada a lista de indicadores definida por Alves *et al.* (2016). Dessa forma, o RepositoryMiner é responsável por: (i) Minerar repositórios de software locais e remotos; (ii) Calcular 22 métricas de software; (iii) Detectar 9 tipos de *code smells*: *Brain Class*, *Brain Method*, *Conditional Complexity*, *Data Class*, *Feature Envy*, *God Class*, *Long Method*, *Refused Parent Bequest* e *Depth of Inheritance Tree*; (iv) Detectar código duplicado utilizando o

[1]Disponível em: https://wiki.dcc.ufba.br/SoftVis/VisminerTD

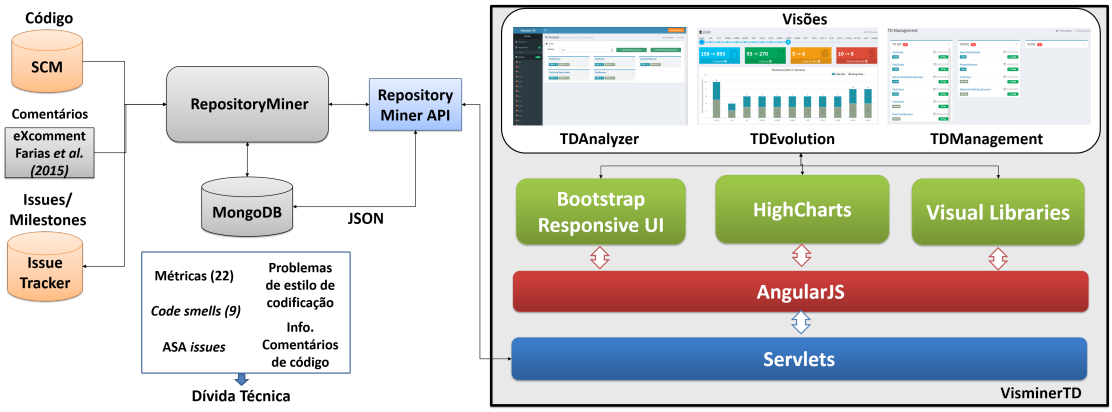

Figura 1. Visão Arquitetural Componente & Conector do VisminerTD

add-on Copy/Paste Detector (CPD) do PMD; (v) Detectar possíveis defeitos em projetos Java através da análise estática do código utilizando a ferramenta FindBugs; e (vi) Detectar comentários em código fonte Java que contenham algum indício da existência de DT bem como os artefatos de software associados a eles [d. F. Farias et al. 2015]. Por meio desses indicadores, o RM é capaz de detectar 8 tipos diferentes de DT. A extração das métricas e *code smells* foi implementada considerando as definições encontradas em Fowler (1999) e Lanza e Marinescu (2006).

VisminerTD foi construído como uma aplicação Java web que utiliza o AngularJS, Bootstrap e o HighCharts para montar suas visões. Sempre que é preciso buscar alguma informação no banco de dados MongoDB, o AngularJS envia uma requisição HTTP para um *servlet*, passando a ação desejada e os parâmetros necessários. O *servlet* Java recebe a requisição, filtra a ação a ser tomada e acessa os dados requisitados. Após isso, o *servlet* se encarrega de enviar o resultado da *query*, em formato JSON, para o AngularJS. Já o Bootstrap ofereceu inúmeros componentes para a criação das *views*, além de permitir que o sistema fosse responsivo. Devido ao alto número de dados e métricas, ficou evidente que seria necessário adicionar gráficos nas visões. Por isso, neste projeto foram utilizados gráficos JavaScript da biblioteca digital HighCharts.

Para apoiar a identificação e monitoramento de itens de DT, o VisminerTD é composto por três módulos complementares: (1) **TDAnalyzer**: oferece uma descrição detalhada de quais tipos e indicadores foram utilizados para indicar os itens de dívida presentes no projeto; (2) **TDEvolution**: permite o entendimento de como o software está evoluindo ao longo do tempo em relação aos itens de dívida identificados; e o (3) **TDManagement**: permite visualizar e monitorar os itens de DT do projeto.

3. Estudo de Viabilidade

Esta seção apresenta o estudo realizado com o objetivo de investigar a viabilidade de uso da ferramenta desenvolvida. Será avaliado se o VisminerTD, através do uso de recursos de visualização de informação, permite realizar atividades de identificação e monitoramento de itens de DT. O projeto escolhido para ser analisado foi o JUnit, que se trata de um *framework open-source* que apoia a criação de testes automatizados na linguagem Java. Ele foi escolhido por ser um projeto muito utilizado na academia e na indústria. Sua versão mais atual está disponível no GitHub e possui 21 versões disponíveis, 137 colaboradores

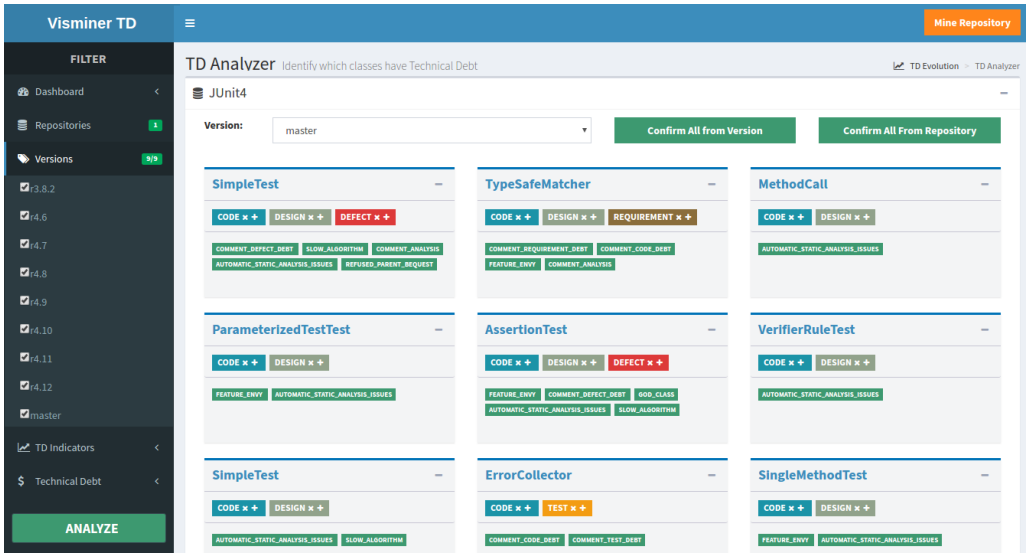

Figura 2. Componente TDAnalyzer

e 2187 *commits*.

O procedimento definido para o estudo simula a realização de uma atividade de verificação da estrutura interna do JUnit com o objetivo de identificar possíveis trechos do projeto que não empregaram boas práticas de desenvolvimento e, dessa forma, podem afetar a manutenibilidade do *framework* (possíveis itens de DT). Essa verificação foi realizada pelos pesquisadores do projeto, com diferentes níveis de experiência. Nos parágrafos a seguir, serão descritas as atividades realizadas no uso de VisminerTD e também algumas informações sobre como a ferramenta apoia cada uma delas.

Inicialmente, o repositório do JUnit foi configurado no módulo RepositoryMiner informando: o nome do repositório, a descrição, o caminho da pasta do projeto, o tipo de repositório, as métricas de software para serem executadas na análise e os tipos de DT para serem identificados. Após o processo de mineração, VisminerTD apresenta no módulo **TDAnalyzer** todos os itens identificados com indícios da presença DT. Esse processo de mineração durou aproximadamente 30 minutos para analisar todas as versões do JUnit.

O próximo passo realizado pelos participantes do estudo foi filtrar as informações retornadas (ver Figura 2). Foram selecionadas 9 versões do JUnit, incluindo a *master*, para terem suas informações exibidas no módulo TDAnalyzer. Feito isso, todos os itens que podiam indicar a presença da DT foram listados como *cards* (“cartões”), sendo possível confirmar ou rejeitar os itens indicados (caso se tratassem de falso positivos). Essa atividade é realizada *card* a *card*, selecionando a opção “+” para confirmar que um determinado item realmente possui aquele tipo de DT ou “x” para descartar a indicação. Também é possível confirmar todos os itens indicados para uma dada versão ou repositório analisado. Para esse estudo, todos os itens foram confirmados.

No total, foram encontrados 2.126 indicações da presença da DT considerando as 9 versões analisadas, resultando em uma média de 236 itens de dívida por versão. Além disso, verificou-se que o artefato com mais indicadores da presença de DT na versão *master* foi o “*AssertionTest*”, no qual foram detectados dois *code smells* (*God Class* e

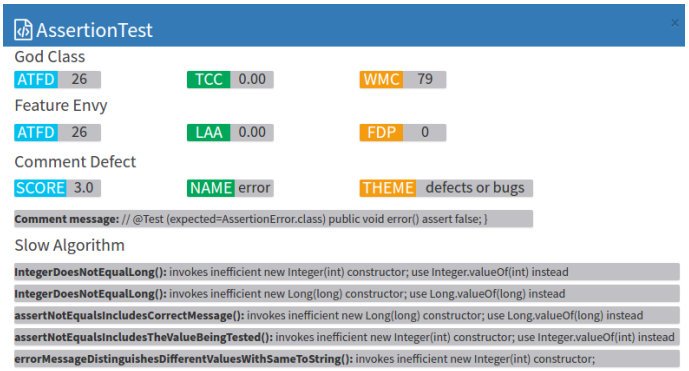

Figura 3. Descrição detalhada dos indicadores do item "AssertionTest"

Feature envy), ASA *issues* do tipo *Slow Algorithm* e um comentário indicando uma dívida de defeito. Para acessar a página com os detalhes dessa classe, pode-se clicar sobre o nome do item (ver Figura 3). Nesta página, são exibidos os valores dos *thresholds* de cada métrica para detecção dos *code smells*, as mensagens de erros de ASA (Automatic Static Analysis) *issues* que foram identificados e as informações sobre os comentários (quando houver).

O próximo passo no uso de VisminerTD é acessar o módulo **TDEvolution** (ver Figura 4), que permite observar como a presença da DT se comporta nas diferentes versões analisadas do JUnit. Essa informação possibilita perceber em quais momentos ocorrem mudanças que podem afetar positivamente ou negativamente a manutenibilidade do projeto. Para isso, TDEvolution utiliza a estratégia diferencial absoluta [Novais et al. 2017], que compara duas versões de software por vez, e é utilizada para apresentar atributos que apareceram ou desapareceram nas diferentes versões. Entre as informações que podem ser comparadas de uma versão para outra, tem-se: número de *commits*, número de classes, quantidade de *indicators* e de itens de DT identificados até o momento. Para ilustrar a evolução da DT ao longo das versões, foi utilizado um gráfico HighCharts chamado *stacked column*, que permite, além da comparação entre duas versões, uma visão global para analisar a evolução da quantidade de itens de dívida em todas as versões do software.

Ao analisar a visão disponibilizada no módulo TDEvolution, verificou-se por

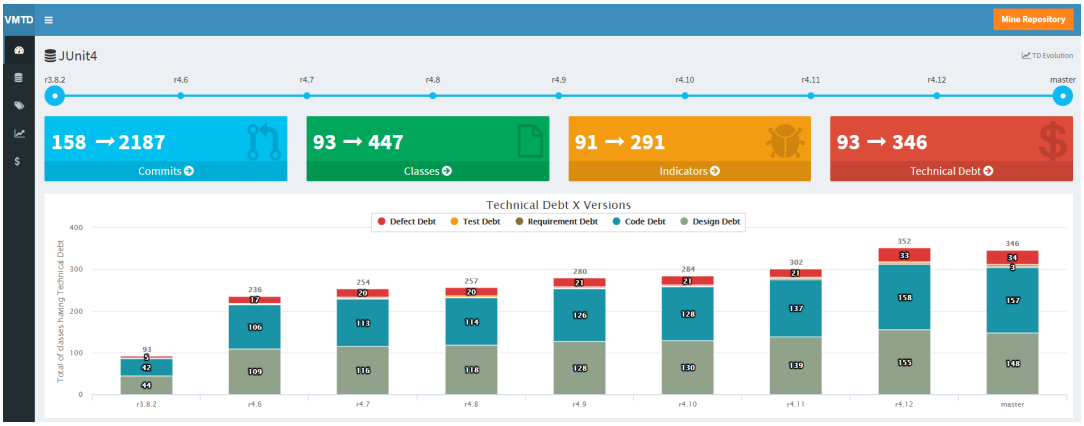

Figura 4. Componente TDEvolution

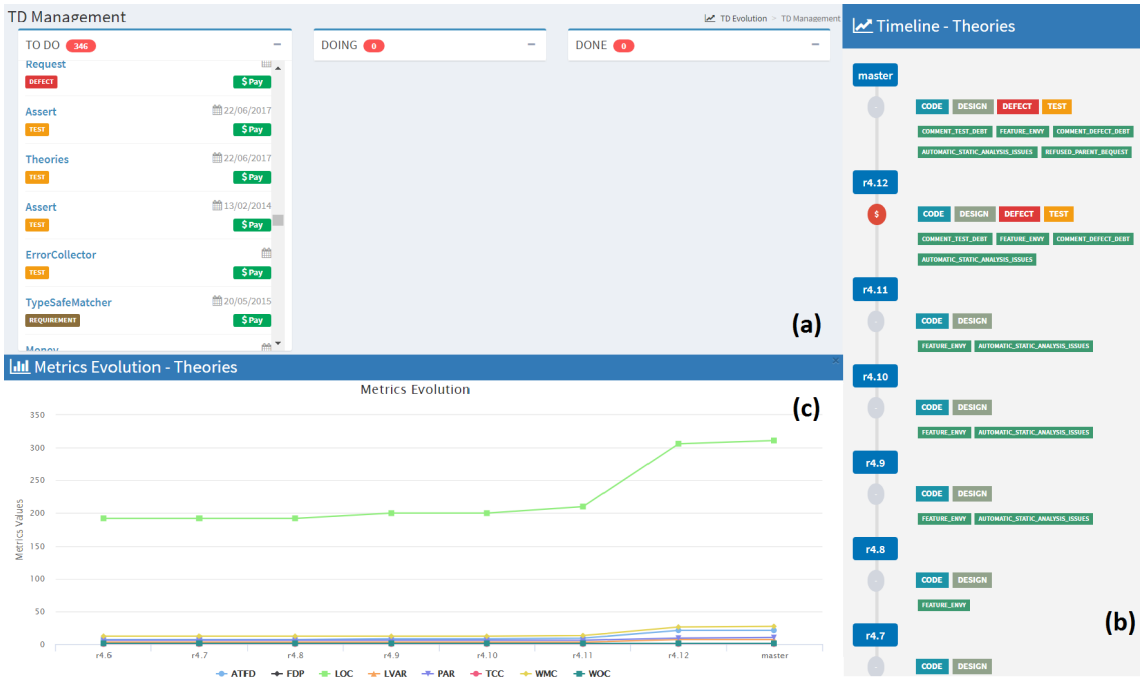

Figura 5. Componente TDManagement

exemplo que, da primeira para a segunda versão do JUnit, houve um aumento significativo do número de classes e de itens de DT (ver *releases* 3.8.2 e 4.6 da Figura 4). A partir da segunda versão, a visão apresentou um comportamento ascendente e gradativo, havendo uma pequena diminuição dos valores na versão *master*. Já na comparação entre versões, o número de *commits* aumentou 13 vezes e o número de classes aumentou 4 vezes, da versão inicial para a final, enquanto que a quantidade de *indicators* e DT teve um aumento gradativo à medida que o software foi evoluindo.

A última etapa do estudo foi a análise do **TDManagement**, que é o módulo responsável por auxiliar no monitoramento da DT (ver Figura 5). Através de um painel principal, o usuário pode visualizar os itens de DT do projeto e quais tipos de dívida estão associados a eles. Esses itens são apresentados utilizando o conceito de Kanban [Silva et al. 2012] com três painéis (TO DO, DOING e DONE).

No painel TO DO são listados todos itens que tiveram as dívidas confirmadas, e cuja a dívida associada ainda não foi paga. Quando o engenheiro de software decide que a dívida deve ser paga, ele pode mover o *card* para o painel DOING. Neste painel estão listados todos os itens de dívida que estão sendo eliminados no momento. Quando o ajuste for finalizado, o item deve ser marcado como pago, movendo o *card* para o painel DONE. Durante esse processo, é possível verificar a quantidade de itens pendentes em cada um dos painéis através de um contador existente ao lado dos nomes TO DO, DOING e DONE.

Neste módulo também é possível obter uma visão detalhada sobre a situação atual de um item de DT, assim como sua evolução ao longo das diferentes versões. Para isso, ao clicar sobre um determinado item, será aberta uma visão *timeline*. Um círculo na cor cinza significa que não houve alteração da versão anterior para a versão em análise; um círculo vermelho indica que alguma dívida foi inserida de uma versão para outra; e a cor verde indica que uma dívida foi removida. Além disso, TDManagement também apresenta um

conjunto de visões contendo informações referentes aos valores das métricas obtidas que levaram à identificação do item de dívida.

Na Figura 5.a tem-se o cenário inicial apresentado em TDManagement, no qual foram mapeados 346 itens de DT no painel TO DO, e nenhum nos painéis DOING e DONE. Entre os *cards* existentes, foi escolhido um da classe Theories para analisar a *timeline* e as visões dos indicadores desse item. A *timeline* da Theories apresentou variações nas duas últimas versões, sendo possível constatar que surgiram dois itens de dívida na versão 4.12, que se mantiveram até a versão *master* (ver Figura 5.b). Por último, na Figura 5.c, é possível observar os valores das métricas associadas ao indicador analisado. Observe que os dados são constantes até a versão *tag* r4.11. Da versão *tag* 4.11 para a *tag* 4.12 houve um aumento no valor de LOC e também nas demais métricas.

Neste estudo verificou-se que o VisminerTD permite identificar itens de dívida através da análise de informações detalhadas e gráficos dos diferentes indicadores de DT, não sendo necessário acesso ao código fonte para identificar os problemas no software. Entretanto, durante a realização do estudo, observou-se a necessidade de melhorias, como a criação de uma funcionalidade que permita a classificação dos itens encontrados de acordo com a prioridade das DT encontradas, entre outras.

4. Trabalhos Relacionados

O mapeamento sistemático realizado por Alves *et al.* (2016) indicou que a identificação e monitoramento de DT por meio de técnicas de visualização de software ainda demanda soluções ferramentais de apoio. Neste mapeamento, foram identificados apenas dois trabalhos que propuseram o uso de técnicas de visualização na atividade de identificação de DT.

O primeiro trabalho relacionado foi DebtFlag [Holvitie and Leppänen 2013], uma ferramenta que permite a marcação do código para "ligar" os itens de dívida com o ponto do projeto em que ele está localizado no código. O segundo trabalho é SonarQube (http://www.sonarqube.org), uma das mais conhecidas ferramentas para inspeção de qualidade de código. Trata-se de um software *open source* capaz de extrair diferentes tipos de relatórios sobre um sistema, como métricas de software, *code smells*, identificação de DT, dependências entre classes, entre outros. O foco dessa ferramenta é apoiar a equipe de desenvolvimento e acompanhar a qualidade do código fonte.

VisminerTD se diferencia em relação às ferramentas citadas. Quando comparada a DebtFlag, por exemplo, tem-se que o processo de identificação dos indicadores de DT e de geração de gráficos é realizado de forma automática e não manualmente. Já em relação ao SonarQube, VisminerTD, além das diferentes métricas coletadas, também combina as informações dos comentários de código fonte para mostrar indícios de DT, permitindo a identificação de diferentes tipos indicadores de DT durante a evolução do software. Além disso, VisminerTD possui recursos relacionados à mineração de repositórios do software e permite a análise de várias versões do software de forma automática.

5. Considerações Finais

VisminerTD apoia a identificação e o monitoramento de itens de dívida utilizando recursos de visualização de software. Atualmente, a ferramenta permite a detecção de 9 *code*

smells, ASA *issues*, problemas de estilo em código Java e 8 tipos de DT (dívida de arquitetura, dívida de construção, dívida de código, dívida de defeito, dívida de documentação, dívida de projeto, dívida de requisitos e dívida de teste). Além disso, ela permite a análise conjunta dos dados provenientes das métricas com as informações extraídas da análise de comentários de código para apoiar a identificação de itens da dívida.

O estudo conduzido forneceu indicativos iniciais sobre a viabilidade de uso do VisminerTD. Os próximos passos desta pesquisa envolvem a realização de estudos experimentais em ambiente acadêmico e na indústria com o objetivo de avaliar de forma mais profunda a solução desenvolvida.

Referências

Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F., and Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70:100 – 121.

Cunningham, W. (1992). The wycash portfolio management system. *SIGPLAN OOPS Mess.*, 4(2):29–30.

d. F. Farias, M. A., d. M. Neto, M. G., d. Silva, A. B., and Spínola, R. O. (2015). A contextualized vocabulary model for identifying technical debt on code comments. In *IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pages 25–32.

Guo, Y., Spínola, R. O., and Seaman, C. (2014). Exploring the costs of technical debt management – a case study. *Empirical Software Engineering*, 1:1–24.

Holvitie, J. and Leppänen, V. (2013). Debtflag: Technical debt management with a development environment integrated tool. In *In 4th MTD*, pages 20–27.

Izurieta, C., Vetrò, A., Zazworka, N., Cai, Y., Seaman, C., and Shull, F. (2012). Organizing the technical debt landscape. In *Third International Workshop on Managing Technical Debt*, MTD '12, pages 23–26, Piscataway, NJ, USA. IEEE Press.

Magnavita, R., Novais, R., Silva, B., and Mendonça, M. (2016). Using evowave for logical coupling analysis of a long-lived software system. In *VEM'2016*, pages 1–8.

Novais, R., Santos, J. A., and Mendonça, M. (2017). Experimentally assessing the combination of multiple visualization strategies for software evolution analysis. *Journal of Systems and Software*, 128:56 – 71.

Novais, R., Simões, P., and Mendonça, M. (2012). Timeline matrix: an on demand view for software evolution analysis. In *Brazilian Work. on Soft. Visualization*, pages 1–8.

Seaman, C. and Guo, Y., editors (2011). *Measuring and Monitoring Technical Debt*, volume 82. Advances in Computers.

Silva, D., Santos, F., and Neto, P. (2012). Os benefícios do uso de kanban na gerência de projetos de manutenção de software. pages 337–347.

Spínola, R. O., Vetrò, A., Zazworka, N., Seaman, C., and Shull, F. (2013). Investigating technical debt folklore: Shedding some light on technical debt opinion. In *4th International Workshop on Managing Technical Debt*, pages 1–7.

Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., and Seaman, C. (2013). A case study on effectively identifying technical debt. In *17th EASE*, pages 42–47. ACM.