Uma abordagem orientada a negócios para identificação de dívidas técnicas a partir de análise estática de código*

Robson Alves da Silva Júnior, Rodrigo Rebouças de Almeida

[1]Universidade Federal da Paraíba (UFPB) - Campus IV
Rio Tinto - PB - Brasil

{robson.alves, rodrigor}@dcx.ufpb.br

Abstract. *Technical debts are shortcuts or pending issues allowed by the development teams to obtain short-term benefits, but these debts can demand more costs and cause negative impacts on the business. The technical debt identification tools based on static code analysis produce only technical metrics to help companies make their decisions. Considering that business and management aspects are one of the main causes of the creation of new debts, this work presents the Tracy-CI solution, which in a context of continuous integration, performs the identification of technical debts in code non-conformities. The tool was deployed in two software development teams, with a total of 17 IT professionals. More than 25 cycles of continuous integration were carried out, each with the result of the impact on the business of technical non-compliance. In addition to making it possible to record the technical debt identified at the end of each cycle.*

Resumo. *Dívidas técnicas são atalhos ou pendências admitidas pelos times de desenvolvimento para obterem benefícios de curto prazo, mas estas dívidas podem demandar mais custos e provocar impactos negativos no negócio. As ferramentas de identificação de dívidas técnicas a partir da análise estática de código produzem apenas métricas técnicas para ajudar na tomada de decisão das empresas. Considerando que aspectos de negócios e gerenciais são uma das principais causas da criação de novas dívidas, este trabalho apresenta a solução Tracy-CI, que em um contexto de integração contínua, realiza a identificação de dívidas técnicas em não conformidades de código. A ferramenta foi implantada em dois times de desenvolvimento de software, com um total de 17 profissionais de TI. Foram executados mais de 25 ciclos de integração contínua, cada uma com o resultado do impacto no negócio da não conformidade técnica. Além de possibilitar de registro da dívida técnica identificada ao final de cada ciclo.*

1. Introdução

Dívida técnica (Cunningham, 1992) é uma metáfora para descrever a situação onde equipes de desenvolvimento de software precisam implementar soluções paliativas convenientes no curto prazo (dívidas), mas que podem causar impacto negativo no futuro (juros

*Trabalho de conclusão de curso, sob orientação do professor Rodrigo Rebouças de Almeida submetido ao Curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação (CCAE) da Universidade Federal da Paraíba, como parte dos requisitos necessários para obtenção do grau de LICENCIADO EM CIÊNCIA DA COMPUTAÇÃO.

das dívidas) (Avgeriou et al., 2016). Por exemplo, um time pode deixar de implementar testes automatizados adequadamente (assumir a dívida de testes) para implementá-los no futuro. No futuro, o custo e esforço para implementar os testes (o custo de pagar a dívida) é mais alto, inclui diversos fatores como complexidade maior do código e concorrência com novas prioridades, por exemplo. Este custo ou impacto adicional são os ”juros”da dívida.

De acordo com Rios et al. (2019) fatores gerenciais e de negócios são uma das principais causas da criação de dívidas técnicas. Prazos apertados para que a empresa atinja metas de negócios, por exemplo, ou mudanças inesperadas de requisitos ao longo do processo de desenvolvimento forçam a equipe de desenvolvimento a tomar atalhos para conseguir entregar os resultados “de alguma forma”, muitas vezes negligenciando a qualidade do produto.

O código é um dos artefatos de desenvolvimento que sofre com diversos tipos de dívidas técnicas, sejam dívidas de testes, de documentação, de qualidade, segurança, entre outras. (Khomykov et al., 2019) apresenta diversas ferramentas que contribuem para identificar dívidas técnicas a partir da análise estática de código, como SonarQube, MIND e FindBugs, por exemplo.

O problema é que estas não conformidades técnicas (Bugs, vulnerabilidades no código, duplicações, entre outros) avaliam o código como se todos eles tivesse a mesma relevância para o negócio. A relação entre não conformidades técnicas e métricas de negócios ainda é pouco explorada no contexto de dívidas técnicas, relacionar essas duas áreas muitas vezes não é algo trivial. Por exemplo, uma nova funcionalidade para vendas em um sistema é desenvolvida sem uma completa cobertura de testes, isso pode gerar um impacto enorme no negócio, como um custo elevado para manter essa funcionalidade, ineficiência em vendas e queda na receita da empresa.

Diante disso, este trabalho tem como objetivo principal **desenvolver uma solução para identificação de dívidas técnicas que afetam o negócio em não conformidades técnicas, a partir de uma ferramenta de análise estática de código e em um contexto de integração contínua**. A solução utilizará o *framework* Tracy proposto por Rebouças de Almeida et al. (2019), que permite identificar o impacto de dívidas técnicas no negócio.

Os trabalhos existentes, relacionados à identificação de dívidas técnicas têm focado na tomada de decisão baseada em métricas técnicas, alguns realizando a identificação direta, em documentações desatualizadas ou testes incompletos, outras através de ferramentas de análise de código, com a captura de não conformidades técnicas (Alves et al., 2016; Rios et al., 2018).

O restante do artigo está organizado em seis seções. A seção 2 apresenta os trabalhos relacionados; a seção 3 apresenta o contexto de gerenciamento de dívidas técnicas com uma perspectiva orientada a negócios; a seção 4 apresenta a metodologia utilizada para a construção da solução; a seção 5 discute a solução deste trabalho, intitulada Tracy-CI; em seguida, a seção 6 apresenta os resultados da implantação da solução em um ambiente real e por fim, a seção 7 apresenta a conclusão e trabalhos futuros.

2. Trabalhos Relacionados

Besker et al. (2019) mostra como as dívidas técnicas são gerenciadas pelos profissionais

da indústria de software, com foco nos fatores que afetam a sua priorização, e indica que a perspectiva de negócios é pouco explorada nas soluções de gerência de dívidas técnicas.

Bellomo et al. (2016) explora a perspectiva de dívidas técnicas em softwares abertos governamentais, e destaca a dificuldade de mapeamento e identificação das dívidas. A pesquisa girou em torno das *issues* do projeto, com a participação de pesquisadores e engenheiros de software, ao fim foram encontradas 109 dívidas técnicas não mapeadas.

O gerenciamento de dívidas técnicas não é algo trivial nas organizações. Através de uma pesquisa feita em 15 organizações e 226 participantes, (Guerlesquin, 2019) destaca que apenas uma pequena parte desse grupo utiliza uma ferramenta para o gerenciamento de dívidas técnicas e um total de 7,2% realizam o monitoramento das dívidas cadastradas.

A partir da coleta e análise de 188 itens de dívidas técnicas de duas empresas de desenvolvimento de software, (Rebouças de Almeida et al., 2019) realizou um comparativo entre a priorização de dívidas com uma abordagem puramente técnica e uma abordagem orientada a negócios. O autor apresenta evidências de que uma abordagem orientada a negócios deixa o processo de tomada de decisão mais alinhado com as expectativas de negócios.

Khomykov et al. (2019) realiza uma pesquisa sistemática sobre a mensuração automática de dívidas técnicas. Softwares livres como SonarQube, MIND e FindBugs realizam a extração de dados em projetos para verificação de não conformidades técnicas. Como resultado as ferramentas suportam automação e estimam apenas o custo de correção técnico.

Com essa perspectiva neste trabalho foi desenvolvida uma aplicação para auxiliar a identificação de dívidas técnicas, através do consumo de não conformidades técnicas pelo SonarQube[1]. Essa aplicação é uma extensão da ferramenta de priorização de dívidas técnicas Tracy-TD (Rebouças de Almeida et al., 2019), que tem como foco a priorização de dívidas técnicas considerando o seu impacto no negócio.

Dentro do escopo da nossa análise de trabalhos existentes, nosso trabalho é o primeiro a utilizar a perspectiva de negócios para auxiliar a tomada de decisão sobre criação ou não de dívidas técnicas a partir de não conformidades obtidas a partir de ferramentas de análise estática de código.

3. Gerência de Dívidas Técnicas Orientada a Negócios

Este trabalho tem como base o *framework* Tracy (Rebouças de Almeida et al., 2019), que utiliza a perspectiva orientada a negócios para a priorização de dívidas técnicas.

A figura 1 apresenta os componentes do *framework* Tracy, que é implementado pela ferramenta Tracy-TD. Basicamente, o framework possui pontos de extensão que são próprios da organização, com estes elementos, utiliza-se de um *Canvas de Priorização* e um *Canvas de Valor do Negócio* para priorizar uma lista de *Dívidas Técnicas* e identificar os impactos potenciais de cada dívida no negócio.

Os pontos de extensão são compostos por: *Ativos de TI*, que são os produtos e

[1]Ferramenta para inspeção contínua de qualidade de software. Disponível em https://www.sonarqube.org/.

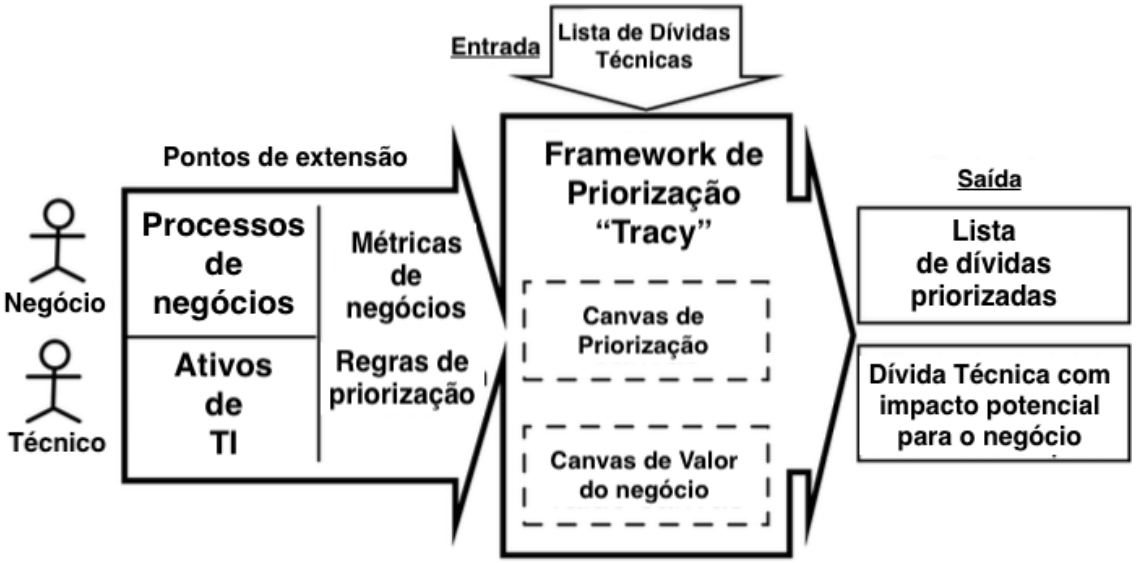

Figura 1. Componentes do framework Tracy.

serviços oferecidos pela empresa ao mercado (em geral os itens presentes em seu portfolio); *Processos de Negócios*, definição das formas como os Ativos de TI geram valor para o negócio. Por exemplo, uma empresa de comércio eletrônico possui um serviço de vendas na internet (um Ativo de TI) que gera valor através da venda de produtos (um processo de negócio). Tanto os Ativos de TI quanto Processos de negócios são associados a *métricas de negócios* (como custo, lucro, oportunidade de mercado, entre outras).

Outro elemento fundamental presente no framework são os *Itens de Configuração*, um sub-componente de um Ativo de TI. *Itens de Configuração* são quaisquer elementos que podem ser alvo de dívidas técnicas, como módulos de sistemas, sistemas, serviços, servidores, etc. Com a relação entre eles, os *Ativos de TI* e os *Processos de Negócios* é possível identificar o impacto das dívidas técnicas.

Neste trabalho foi necessário estabelecer a relação entre o que chamamos *Componentes de código* (classes, pacotes ou qualquer outro arquivo presente no código fonte) e os *Itens de Configuração*, para identificarmos as áreas de código que se relacionam com os elementos descritos no framework Tracy.

4. Metodologia

Este trabalho está estruturado segundo uma *Technical Action Research* (Wieringa, 2014), que permite a avaliação de um artefato experimental em um ambiente real. O trabalho foi conduzido nas seis etapas a seguir, conforme a figura 2.

1) Levantamento de requisitos técnicos para uma solução;

Alguns requisitos técnicos são necessários para o desenvolvimento de uma solução para avaliar se o impacto no negócio das não conformidades identificadas a partir da análise estática do código melhoram a decisão sobre a criação ou não de dívidas técnicas.

Inicialmente é necessário ter uma ferramenta de análise estática de código, que irá fornecer métricas após a análise do projeto, como: bugs; vulnerabilidades; "bad smells"

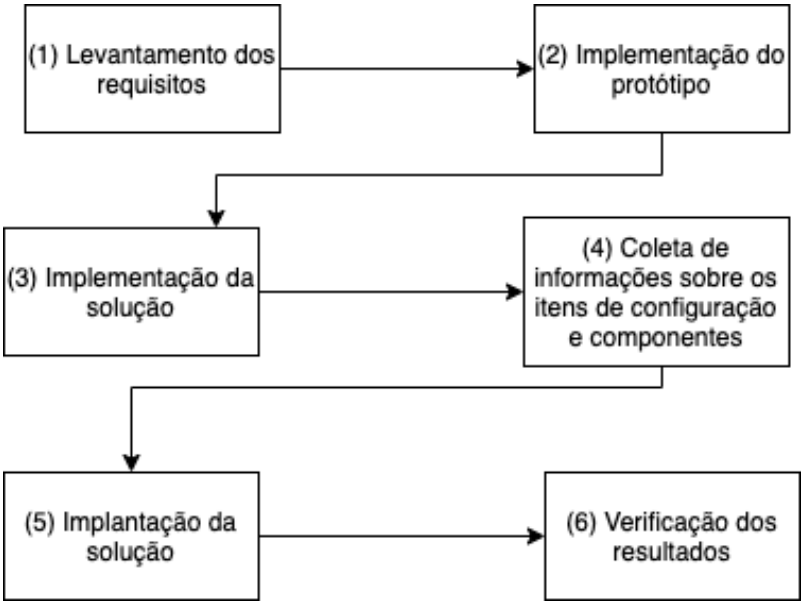

Figura 2. Metodologia.

no código; duplicações no código e cobertura de testes. A partir disso foi seleciona uma ferramenta de análise estática de código e uma métrica de não conformidade técnica para este trabalho.

Para que a identificação aconteça de forma automática e contínua os projetos deve ocorrer em um contexto de integração contínua, para que a cada ciclo de desenvolvimento possa ser possível consultar o impacto no negócio das não conformidades.

2) Implementação de protótipo

A implementação de um protótipo para validar se a solução era viável foi necessário. O desenvolvimento do mesmo foi com o *framework* Express, utilizando a linguagem de programação Javascript.

Para mapeamento das não conformidades técnicas, criou-se uma solução de mapeamento dos componentes de código com os *Itens de Configuração* indicados no *framework* Tracy e implementados na ferramenta Tracy-TD. A validação do protótipo foi realizada com as não conformidades técnicas identificadas no próprio projeto da Tracy-TD.

3) Implementação da solução Uma vez validada a solução, foi realizado o desenvolvimento da solução final, para que a solução consiga se comunicar com serviços de integração contínua, consumir as não conformidades técnicas de uma ferramenta de análise estática de código e o impacto do negócio da Tracy-TD.

4) Coleta de informações sobre os *Itens de Configuração* e componentes

Para ter um maior entendimento em relação as métricas de negócios do negócio, é necessário realizar reuniões com os líderes das equipes, para assim entender os módulos existentes, métricas de negócios e o processo de integração contínua utilizado.

A partir do entendimento e análise dos módulos, foram coletadas informações

técnicas sobre os sistemas e as métricas de negócios relevantes dos projetos, para posteriormente realizar a relação entre *Itens de Configuração* e componentes de código.

5) Implantação da solução

O processo de implantação da solução deve ocorrer dentro do ciclo de integração contínua dos times, para que a captura de não conformidades técnicas aconteça de forma automática. A solução deve ser capaz de consumir as não conformidades técnicas e verificar o impacto no negócio a cada novo ciclo de integração.

6) Verificação dos resultados

A verificação dos resultados foi feita através da de uma entrevista com todos os envolvidos no processo, a fim de compreender se o uso das informações sobre o impacto no negócio do resultado da análise estática de código melhora ou não a decisão sobre criação de dívidas técnicas. Além de colher *feedbacks* sobre o uso da solução em um ambiente de desenvolvimento.

5. Solução Tracy-CI

Existem diversas ferramentas que realizam a identificação automática de dívidas técnicas através de uma análise estática de código, mas nenhuma mede o impacto de uma não conformidade técnica no negócio. A ferramenta Tracy-TD traz uma perspectiva orientada a negócios para a priorização de dívidas técncias, mas não apresenta uma relação direta de não conformidades técnicas com métricas de negócios. Dessa forma, a solução deste trabalho, a Tracy-CI, é capaz de extrair as não conformidades técnicas geradas pelo SonarQube, e a partir disso informar ao usuário o impacto da não conformidade técnica no negócio.

A figura 3 apresenta o fluxo de como a solução se comporta. As setas indicam o fluxo de informações entre os componentes da solução. Inicialmente, a Tracy-CI deve estar conectada ao SonarQube em um ambiente de integração contínua, para captura automática de métricas e não conformidade técnicas. Para isso, o SonarQube deve ser configurado para enviar um *payload* de informações da análise através de um *hook* para a Tracy-CI.

Para identificação do impacto no negócio a partir das não conformidades técnicas é necessário mapear um conjunto pré definido de pacotes ou classes do projeto "Componentes de Código" para um "Item de Configuração".

O processo se inicia a partir do momento em que um desenvolvedor realiza um *pull request* no projeto do GitLab ou GitHub. Após isso, uma ferramenta de integração contínua é invocada, realizando uma *pipeline* de atividades. Uma dessas atividades é a inspeção contínua da qualidade do código feita pelo SonarQube.

O código solicitado no *pull request* será analisado pelo SonarQube, que irá produzir métricas a partir da análise estática do código fonte. Algumas métricas são detecção de bugs, vulnerabilidades e cobertura de testes.

Em seguida, o *hook* do SonarQube é ativado, enviando as métricas para a Tracy-CI, que realiza o mapeamento dos componentes de código afetados pelo *pull request* com os *Itens de Configuração* de Tracy-TD. Em seguida, Tracy-CI envia as não conformidades para a ferramenta Tracy-TD, que identifica o impacto do código presente no *pull request*

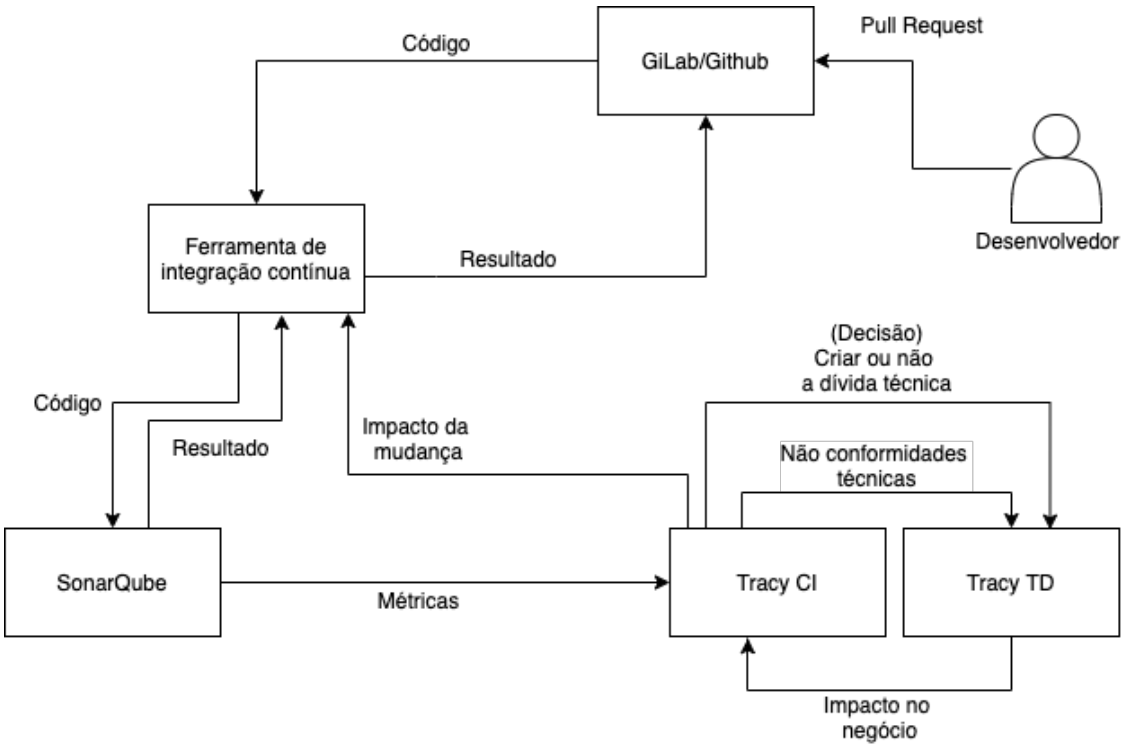

Figura 3. Elementos da solução Tracy-CI e seus relacionamentos.

para o negócio dado um conjunto de métricas cadastradas.

Cada novo ciclo de desenvolvimento, desde o *pull request*, envio de não conformidades técnicas do SonarQube para a Tracy-CI, análise das não conformidades e métricas de negócios é intitulado neste trabalho como *build*.

Ao fim de cada ciclo a Tracy-CI registra um novo *build* dentro da Tracy-TD, que pode ser posteriormente consultado. Dentro de cada *build* é possível visualizar os *Itens de Configuração* afetados, assim como os *Ativos de TI* afetados e o impacto no negócio, além de uma opção para criação de uma nova dívida técnica.

6. Resultados

A extensão Tracy-CI da ferramenta Tracy-TD foi avaliada através da implantação da ferramenta em dois times de desenvolvimento de software, intitulados neste trabalho como ”Equipe A”e ”Equipe B”.

A Equipe A é do próprio contexto de desenvolvimento da ferramenta Tracy-TD. A equipe é bem estruturada e se baseia em metodologias ágeis de desenvolvimento. É composta por cinco pessoas, sendo uma da área de negócios e quatro desenvolvedores.

O ambiente de versionamento de código, além do processo de integração e entrega contínua é realizado com a ferramenta Github, a equipe também utiliza o SonarQube para a realização da análise estática do código. O código avaliado é desenvolvido em Spring Boot, um *framework* feito com a linguagem de programação Java para construção de aplicações de forma rápida. O projeto tem ao todo 18 *Itens de Configuração* que representam banco de dados, sistemas e módulos de sistemas.

A Equipe B é de uma empresa que atua no processamento de transações de pagamentos, a empresa já existe a mais de 20 anos no mercado. A equipe desenvolve software utilizando um processo de desenvolvimento próprio, baseado em práticas ágeis e com integração contínua. Os sistemas desenvolvidos processam centenas de transações por minuto. A equipe é composta por doze pessoas, sendo nove desenvolvedores, um membro da área de negócios, um líder do time e um arquiteto de software.

O versionamento do código é realizado com o GitLab, além disso a equipe utiliza a ferramenta TeamCity[2] para a realização da integração e entrega contínua e o SonarQube para a análise estático do código. Os projetos analisados são desenvolvidos com os *frameworks* Spring Boot e Angular, utilizando as linguagens de programação Java e TypeScript respectivamente.

A partir da entendimento do processo de integração contínua dos times, além das ferramentas utilizadas de análise estática de código, foi decidido utilizar a métrica Cobertura de Testes fornecida pelo SonarQube para execução da Tracy-CI. Após isso, um *hook* foi configurado no SonarQube das equipes para para enviar as informações da análise para a Tracy-CI.

Com o SonarQube configurado, o próximo passo é a configuração da ferramenta de integração contínua. Essa passo foi realizado primeiramente com a Equipe A. Na pipeline de atividades de integração contínua do GitHub foi adicionado uma nova tarefa, responsável por disparar uma ação para a Tracy-CI verificar as não conformidades técnicas que afetam alguma métrica de negócio relevante para a empresa.

A ferramenta de integração contínua da Equipe B é o TeamCity, dentro desse contexto, o passo de configurar o retorno da Tracy-CI para o TeamCity não foi necessário, pois a equipe já tinha configurado uma *pipeline* para falha do *build* caso a métrica de cobertura de testes estivesse abaixo do esperado. O que não afetou o trabalho, já que o processo de verificação das informações do impacto das não conformidades técnicas é visualizado dentro da interface da Tracy-CI.

O processo de implantação foi realizado com exito nas duas equipes, contemplando todo fluxo para identificação do impacto no negócio a partir de novas não conformidades técnicas. Com a facilidade de visualizar o impacto que um determinado código com uma não conformidade técnica tem na empresa, as equipes começaram a identificar e criar novas dívidas técnicas através da própria interface da Tracy-CI.

Durante o período de execução na Equipe A, todos os componentes do projeto foram mapeados para os *Itens de Configuração* correspondentes, ao todo foram 28 *Itens de Configuração* mapeados para 15 componentes do projeto em Spring Boot, entre eles módulos de autenticação, banco de dados, serviços externos, entre outros.

A Equipe B teve ao todo 150 componentes mapeados em seu projeto, os componentes foram relacionados com os *Itens de Configuração* associados. Essa equipe teve em sua base *Itens de Configuração* associados entre si.

Foi obtido um total de 15 *builds* cadastrados para a Equipe A e 13 *builds* para a Equipe B. Cada *build* com um número diferente de *Itens de Configuração* afetados.

[2]Serviço de gerenciamento de compilação e integração contínua. Disponível em https://www.jetbrains.com/teamcity/

Alguns *Itens de Configuração* por sua vez possuíam dependências com outros elementos, o modulo de autenticação por exemplo, afeta o *Item de Configuração* relacionado a permissões.

Para verificação dos resultados, foram entrevistados dois profissionais. A entrevista foi realizada através de video conferência, conduzida pelo segundo autor. O primeiro profissional é coordenador de tecnologia com mais de 18 anos de experiência no mercado, o segundo é líder do time e líder técnico, além de possuir mais de 10 anos de experiência no mercado.

Foram realizadas as seguintes perguntas:

1) Quais os aspectos positivos e negativos da solução? Os aspectos positivos das respostas foram:

P1: ”Qualquer coisa que traga informação do negócio nas decisões técnicas é importante. O que chama a atenção é a ligação de um build com a visão de negócio”

P1: ”A cobertura de testes, por exemplo, 20 %, é bom ou é ruim? 80 % é bom ou é ruim? Isso depende da criticidade dos testes para o negócio.”

P2: ”A informação sobre o negócio ajuda na tomada de decisão sobre aceitar ou não o build”

P2: ”Um aspecto positivo é a facilidade de registrar uma dívida técnica rapidamente, a partir de um build.”

2) Quais as dificuldades encontradas ao longo do processo?

Em relação as dificuldades e aspectos negativos os profissionais responderam:

P1: ”Dá trabalho. Capturar a visão de negócio é difícil”

P1: ”Se a empresa possui os processos bem definidos, bem estruturados, fica mais fácil. Se não tiver, fica muito complexo. Mas esta necessidade não deixa de gerar oportunidade de estruturar os processos.”

P2: ”Elencar os itens de configuração é difícil. Encontrar a granularidade correta não é trivial.”

P2: ”Precisamos organizar a rotina do time para integrar uma solução como esta.”

A partir disso podemos perceber que a Tracy-CI em um contexto de integração contínua, contribui para a identificação de dívidas técnicas a partir de um build, mas a identificação dos processos de negócio e itens de configurações ainda ainda não é algo trivial para os times de desenvolvimento.

7. Conclusão e trabalhos futuros

Foi apresentado neste trabalho a Tracy-CI, uma extensão da Tracy-TD, que tem como objetivo a identificação de dívidas técnicas a partir de uma análise estática de código, através de uma perspectiva orientada a negócios em um contexto de integração contínua.

A ferramenta foi concebida e implantada no processo de integração contínua de duas equipes, sendo a primeira a própria equipe de desenvolvimento de Tracy-TD, e uma segunda equipe de uma empresa que atua no processamento de transações financeiras.

Ao todo, mais de 25 builds foram cadastrados automaticamente na extensão Tracy-CI e cada build possui informações sobre o impacto no negócio dos *pull requests*. Também é possível registrar uma dívida técnica a partir de um *pull request*.

Como trabalhos futuros, propõe-se o desenvolvimento da Tracy-CI para abranger diferentes tipos de métricas geradas pelo SonarQube, além de analisar a possibilidade da generalização do consumo de não conformidades técnicas para qualquer ferramenta de análise estática de código, uma vez que várias ferramentas são utilizadas no mercado.

Referências

Nicolli S.R. 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. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2015.10.008. URL http://www.sciencedirect.com/science/article/pii/S0950584915001743.

Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. Managing technical debt in software engineering (dagstuhl seminar 16162). Dagstuhl Reports, 6, 01 2016. doi: 10.4230/DagRep.6.4.110.

S. Bellomo, R. L. Nord, I. Ozkaya, and M. Popeck. Got technical debt? surfacing elusive technical debt in issue trackers. In 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pages 327–338, May 2016. doi: 10.1109/MSR.2016.041.

T. Besker, A. Martini, and J. Bosch. Technical debt triage in backlog management. In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), May 2019. doi: 10.1109/TechDebt.2019.00010.

Ward Cunningham. The wycash portfolio management system. SIGPLAN OOPS Mess., 4(2):29–30, December 1992. ISSN 1055-6400. doi: 10.1145/157710.157715.

V. Guerlesquin. How (not) to remove technical debt in testing environments. In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), pages 1–2, May 2019. doi: 10.1109/TechDebt.2019.00008.

Ilya Khomykov, Zufar Makhmutov, Ruzilya Mirgalimova, and Alberto Sillitti. Automated measurement of technical debt: A systematic literature review. pages 95–106, 01 2019. doi: 10.5220/0007675900950106.

R. Rebouças de Almeida, C. Treude, and U. Kulesza. Tracy: A business-driven technical debt prioritization framework. In Proc. of the Int'l. Conf. on Software Maintenance and Evolution - ICSME'19, 2019.

N. Rios, R. Oliveira Spínola, M. Mendonça, and C. Seaman. Supporting analysis of technical debt causes and effects with cross-company probabilistic cause-effect diagrams. In 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), pages 3–12, May 2019. doi: 10.1109/TechDebt.2019.00009.

Nicolli Rios, 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 and Software Technology, 102: 117 – 145, 2018. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2018.05.010. URL http://www.sciencedirect.com/science/article/pii/S0950584918300946.

Roel J. Wieringa. Design Science Methodology for Information Systems and Software Engineering. Springer-Verlag Berlin Heidelberg, 1 edition, 2014.