Implementação de um plugin no IntellIJ para Tracy-TD que identifica impactos dos artefatos no negócio

Raimundo M. G. Ludgério[1], Rodrigo R. de Almeida[1]

[1] Departamento de Ciências Exatas (DCX) - Universidade Federal da Paraíba (UFPB)
Caixa Postal 58297-000 – Rio Tinto – PB – Brasil

[1]{raimundo.marcos, rodrigor}@dcx.ufpb.br

Resumo. *Dívidas técnicas são ocorrências que podem impactar negativamente o negócio. As dívidas geradas conscientemente, são chamadas de auto-admitidas e podem ser gerenciadas. O Tracy-TD é uma ferramenta de gerenciamento de dívidas técnicas auto-admitidas, ele disponibiliza a priorização com base nos componentes que was dívidas estão associadas. A partir disso, é possível perceber quais componentes são mais importantes para o negócios, portanto faz-se necessário uma ferramenta que permita a percepção de componentes mais importantes para o negócio. Portanto, o presente trabalho apresenta um plugin para IntelliJ vinculado ao Tracy-TD que permite a visualização dos componentes mais importantes para o negócio enquanto eles estão sendo desenvolvidos.*

1. Introdução

As dívidas técnicas são atalhos ou pendências admitidas pelos times de desenvolvimento para obterem benefícios de curto prazo, algumas podem demandar mais custos e provocar impactos negativos no negócio (ALMEIDA; TREUDE; KULESZA, 2019). Existem diversos fatores que contribuem para a criação delas, os mais comuns são os atrasos nas entregas e imaturidade dos artefatos desenvolvidos. (WEHAIBI; SHIHAB; GUERROUJ, 2016).

Essa área de pesquisa vem sendo explorada há mais de vinte anos com o intuito de minimizar os prejuízos provocados. Além disso, as dívidas técnicas podem ser identificadas de forma consciente, chamadas de "auto-admitidas", cujo os profissionais têm ciência dela e tentam agir de acordo com seus impactos ou de forma inconsciente (POTDAR; SHIHAB, 2014).

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.

Como consequência do impacto que as dívidas técnicas provocam nas organizações, é necessário processos de gerenciamento para que sejam tratadas e administradas auxiliando na tomada de decisão sobre a eliminação ou o momento adequado para que as ações ocorram (GRIFFITH et al., 2014). Para o gerenciamento das dívidas técnicas foi proposto por Almeida et. al. 2018 o Tracy, um framework para a gerência de dívidas técnicas orientadas a negócios, em seguida a ferramenta Tracy-TD que implementa o framework. (ALMEIDA; TREUDE; KULESZA, 2019).

1.1 Tracy-TD

O Tracy-TD é uma ferramenta para gerência de dívidas técnicas orientadas ao negócio, ela permite a gerência dos artefatos pertencentes ao projeto, como produto, componentes, key features e dívidas técnicas. Além disso, o Tracy-TD permite a priorização das dívidas técnicas de acordo com métricas definidas na ferramenta, e possui integração com o Redmine e SonarQube. (ALMEIDA; TREUDE; KULESZA, 2019)

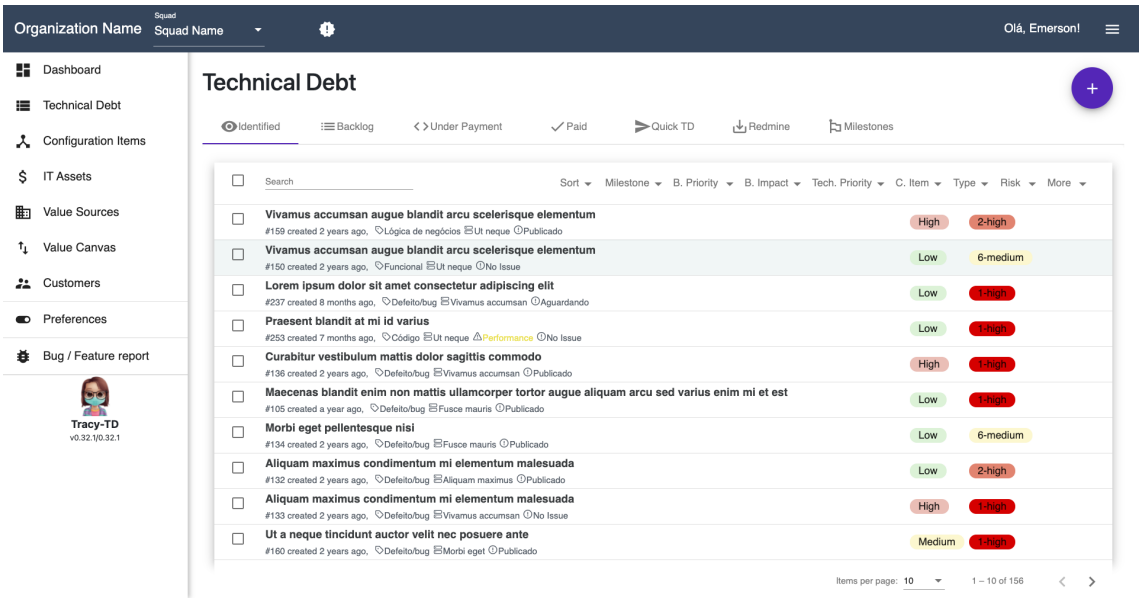

Figura 1: Tela de Listagem de dívidas técnicas do Tracy-TD

O Tracy-TD oferece algumas funcionalidades relacionadas à gerência de dívidas técnicas como criação, edição, priorização e listagem (Figura 1). Ao cadastrar uma nova dívida, é necessário informar qual é o componente que a dívida está associada. Isso permite a classificação de acordo com os componentes mais importantes para o negócio.

Vale ressaltar, também, que dentro de um componente existem artefatos que são desenvolvidos para garantir o seu funcionamento.

O nível de criticidade dos artefatos é medido com base na relação entre o componente e o negócio. As dívidas técnicas relacionadas a componentes mais importantes para o negócio possuem prioridade mais alta, enquanto as dívidas relacionadas a componentes menos importantes para o negócio, possuem prioridade mais baixa. O objetivo deste trabalho é apresentar um plugin que mostra a classificação de prioridade dos artefatos de código para a pessoa desenvolvedora de software. Com isso, espera-se que a ciência sobre a criticidade do artefato que está sendo implementado influencie na qualidade do código que está sendo produzido.

1.2 Proposta do plugin

O plugin[1] oferece à pessoa desenvolvedora de software a percepção prévia dos impactos que os artefatos de código causam no negócio. Para isso, o plugin permite que a interface do IntelliJ classifique os artefatos com cores, de acordo com suas criticidades para o negócio.

As cores dos artefatos são alteradas de acordo com o nível de prioridade e impacto no negócio: os artefatos que possuírem a cor vermelha têm prioridade crítica, possuem nível máximo de prioridade. Os artefatos de cor amarela são aqueles de prioridade alta, em azul são aqueles de prioridade média e em verde, prioridade baixa.

2. Metodologia

2.1 Diferença entre IntellIJ e VSCode

Para o desenvolvimento do plugin foram consideradas duas plataformas: o IntelliJ e o VSCode. A decisão de qual destas estão mais adequadas para o projeto foi tomada com base em duas provas de conceito (PoC), uma no VSCode e uma no IntelIJ, com a finalidade de mensurar o grau de dificuldade para desenvolvimento de plugins apresentado. As PoCs consistiram em desenvolver um plugin teste que fizesse

[1] https://github.com/tracy-td/tracy-intelij

comunicação com um serviço externo e alterasse a interface do ambiente de acordo com a resposta obtida.

No VSCode, a prova de conceito foi desenvolvida em JavaScript, linguagem de programação utilizada para desenvolvimento de extensões no ambiente. A biblioteca utilizada para foi a Axios[2], ela é responsável por efetuar requisições para serviços através do protocolo HTTP. A comunicação foi realizada com sucesso, porém, foi percebido um grau de complexidade alto na alteração da interface.

No IntelIJ, o processo foi o mesmo. Com a possibilidade de utilizar a linguagem Java ou Kotlin, pois o ambiente dá suporte às duas. Construiu-se, portanto, um plugin teste, feito em Java, que efetuava uma comunicação com um serviço externo e alterava a interface do ambiente. A comunicação foi feita utilizando a classe HttpURLConnection[3] do Java. A comunicação foi realizada com sucesso e a alteração da interface se mostrou menos complexa, em relação ao VSCode.

Portanto, o ambiente de desenvolvimento que está mais alinhado com a proposta para o projeto, de alterar a interface do sistema, o IntelIJ se mostrou mais eficiente.

2.3 Implementação

Para o desenvolvimento do plugin utilizou-se a linguagem de programação Java com a comunicação feita pela biblioteca Retrofit[4], que permite conexões seguras entre serviços e é responsável pela comunicação entre o plugin e o módulo de classificação dos artefatos do Tracy-TD. Além disso, o plugin faz a verificação periódica dos artefatos, para que haja um controle em artefatos criados ou alterados, de acordo com a ocorrência de dívidas técnicas. Ou seja, o monitoramento do nível de criticidade dos artefatos é feito em tempo real e caso haja alteração na prioridade, o artefato mudará de cor.

2.2 Plugin para IntellIJ

O processo de desenvolvimento de plugin para IntellIJ é análogo a diversos projetos em Java. Para desenvolvimento do plugin, são necessários arquivos de configuração e

---

[2] https://axios-http.com/docs/intro

[3] https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html

[4] https://square.github.io/retrofit/

gerenciamento de dependência. Além disso, a JetBrains, desenvolvedora do IntelIJ, dá suporte consistente e dois caminhos para desenvolvimento de plugin para a IDE, tanto utilizando Java quanto Kotlin. A pessoa desenvolvedora pode optar por criar um repositório a partir de um template disponibilizado pela JetBrains, com os arquivos de configuração, gerenciamento de dependências e estrutura base para criação ou, também, criar um novo projeto plugin no IntelliJ, dessa forma todos os arquivos necessários serão gerados.

2.2 Estrutura do plugin

A estrutura base de plugin para IntelliJ é semelhante a diversos projetos em Java. Ela consiste em arquivos de configuração, gerenciamento de dependências e classes principais que contém todos os recursos necessários para que o plugin funcione como o esperado. Nesse caso, utilizou-se o Gradle para gerenciamento de dependências, o arquivo de configuração principal utiliza o XML e as configurações são feitas em Kotlin.

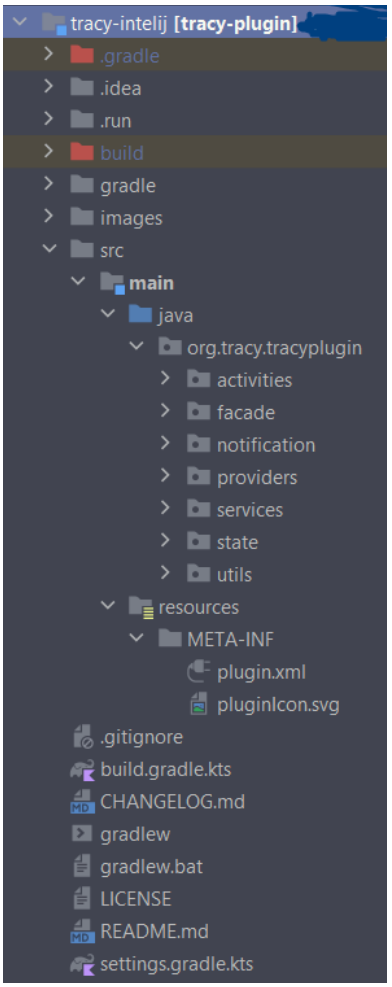

Figura 2: Estrutura base do plugin

Na Figura 2 é possível visualizar a estrutura do plugin, é possível perceber como a estrutura dos arquivos é feita, com diretórios e pacotes que fazem o plugin funcionar. Os arquivos mais importantes são o "plugin.xml" nele encontram-se todas as informações do plugin como o nome, autor, contato, descrição e todas as informações relacionadas ao plugin. Além disso, existe o arquivo de configurações em kotlin, nele é feito a instalação das dependências, informações sobre versionamento, ferramentas utilizadas no plugin e as informações de publicação do plugin no marketplace da Jetbrains[5].

Além disso, na estrutura do plugin é possível visualizar os pacotes onde estão as classes que fazem o plugin funcionar. No pacote "activities" estão as classes responsáveis pelas atividades do plugin como a função que é chamada quando o plugin é instalado a primeira vez, quando ele é carregado na inicialização da IDE ou uma nova versão é atualizada.

O pacote "facade" é responsável pela comunicação com o Tracy-TD, nele estão contidas as classes de configuração do Retrofit. O pacote "notification" é onde estão configuradas as notificações do plugin: mensagem de instalado com sucesso, mensagens de erros e entre outras.

No pacote "providers" estão as classes que fazem o mapeamento dos artefatos presentes no projeto, ela é responsável por alterar as cores dos arquivos.

O pacote "services" é responsável pelas funções do plugin, está encarregado de distribuir as responsabilidades e chamar os métodos responsáveis por cada parte da execução do plugin.

No pacote "state" estão os estados, ele é responsável por salvar qual estado o plugin se encontra e é responsável por evitar que os dados sejam apagados quando a IDE é finalizada. Por fim, o pacote "utils" que são todas as funções que são comuns em várias partes do código.

[5] https://plugins.jetbrains.com/

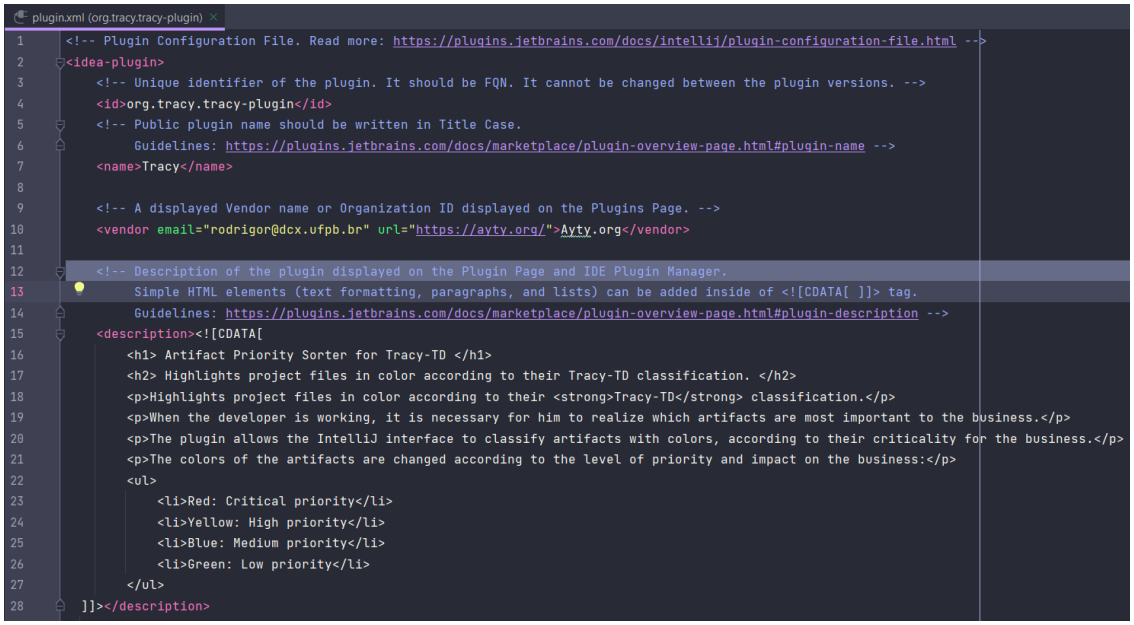

Figura 3: Conteúdo do arquivo plugin.xml

A Figura 3 mostra o arquivo de configuração “plugin.xml”, onde é possível perceber quais informações estão inseridas no plugin. É neste arquivo que encontra-se as informações como o nome do plugin, contato com o desenvolvedor, site do desenvolvedor e uma descrição sobre como o plugin funciona e para que ele serve. Essa informação é exibida na página inicial do plugin no site do Marketplace da Jetbrains.

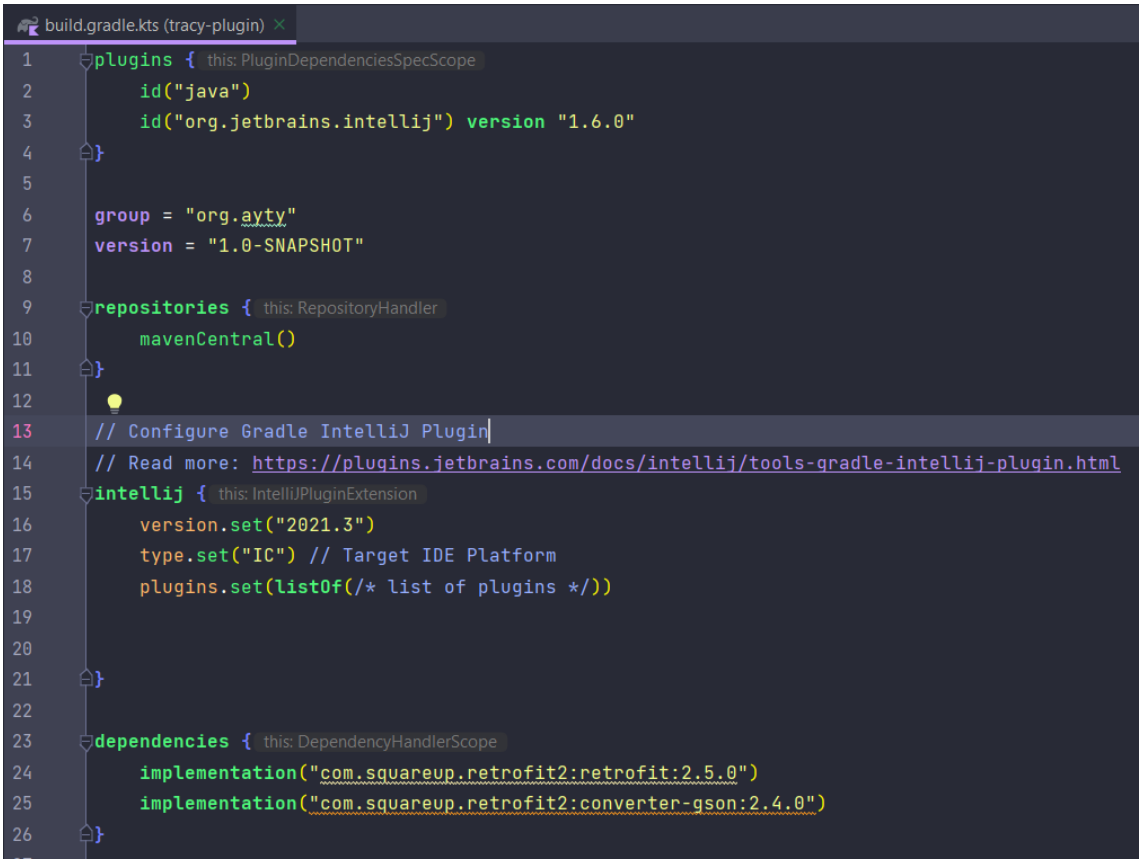

Figura 4: Conteúdo do arquivo build.gradle.kts

A Figura 4 exibe as informações contidas no arquivo “build.gradle.kts” esse arquivo é responsável pelas configurações do gerenciador de dependências e informações relacionadas a versão do plugin, a partir de qual versão do IntellIJ o plugin funciona e quais dependências são utilizadas no plugin, como pode-se perceber nas linhas 23 e 24 estão relacionadas ao retrofit, biblioteca que faz comunicação com o Tracy-TD.

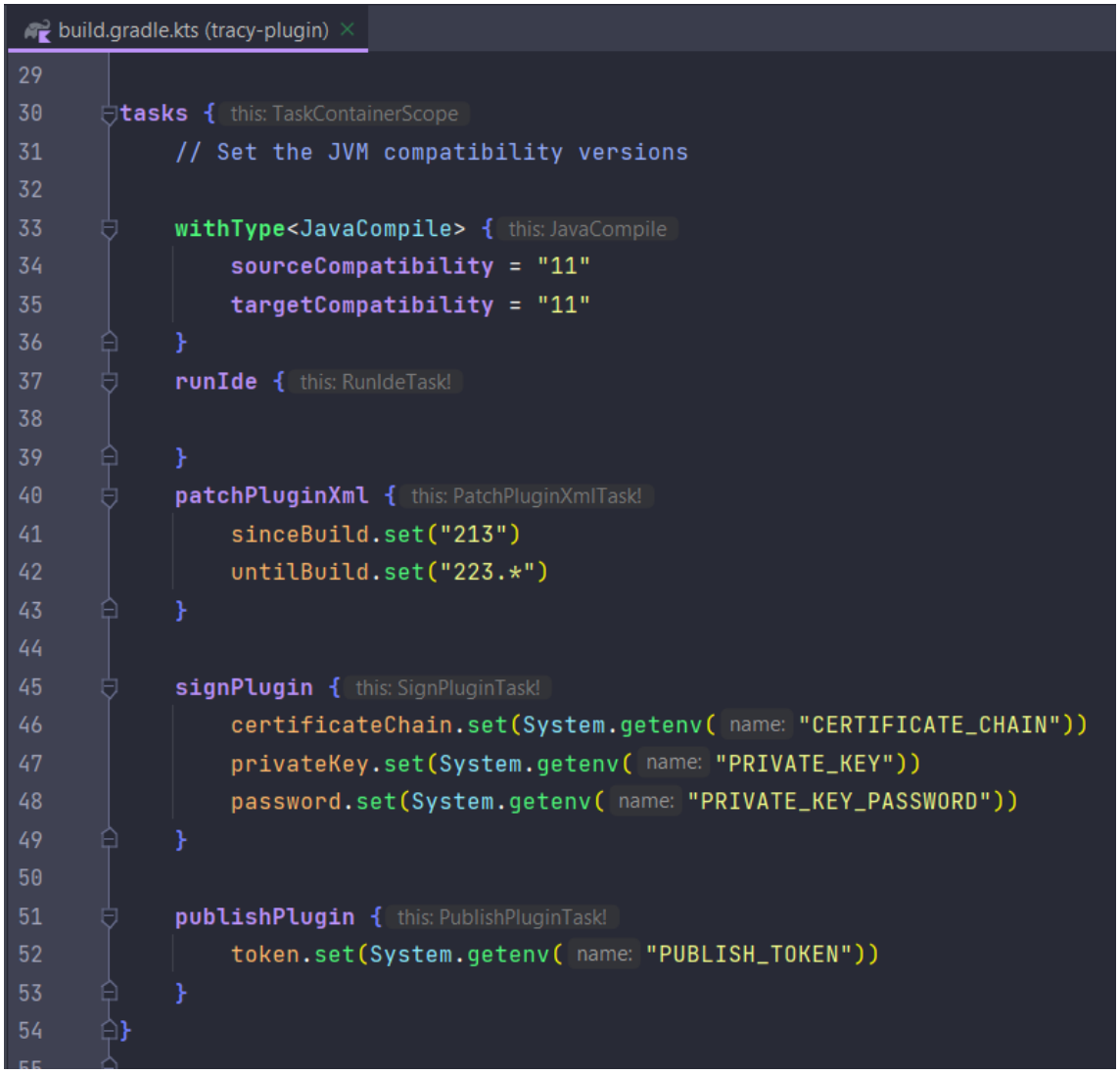

Figura 5: Versão da linguagem e publicação do plugin

Na Figura 5 é possível analisar as informações relacionadas a qual versão da linguagem o plugin está implementado e credenciais para publicação do plugin no marketplace da Jetbrains.

3. Resultados

3.1 Representação do plugin

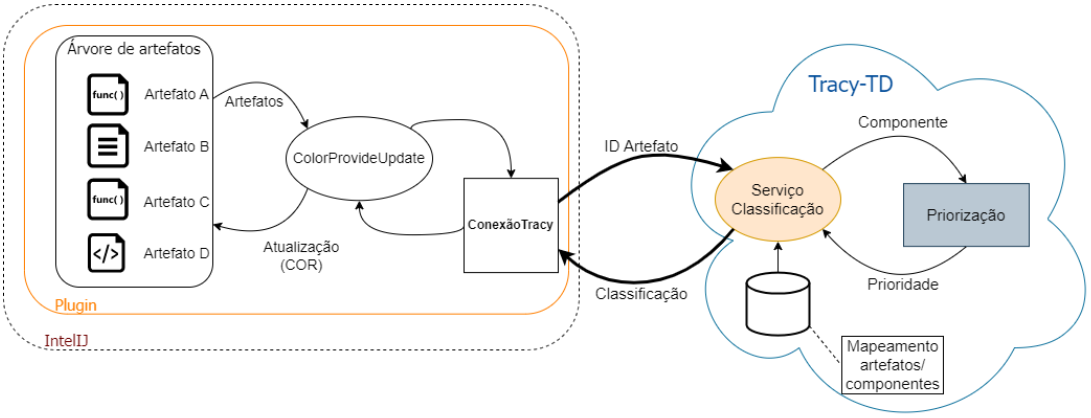

Figura 2: Representação gráfica do funcionamento do plugin

Na Figura 2 é possível observar o diagrama arquitetural do plugin e a comunicação com o Tracy-TD. O plugin faz uma varredura de todos os artefatos do projeto utilizando a classe ColorProviderUpdate, essa classe envia os artefatos que fazem parte do negócio para a classe ConexãoTracy, essa classe envia o ID do artefato para o serviço de classificação em Tracy-TD que vai avaliar o nível de prioridade e retorna a sua classificação. Em seguida a classificação do artefato é retornada e a alteração da interface da IDE é alterada.

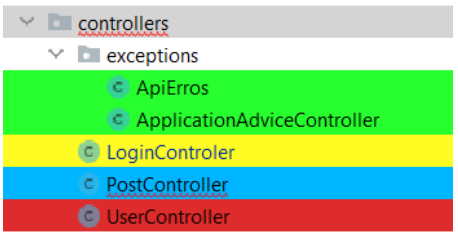

Figura 3: Estrutura de arquivos classificados

Após a verificação de nível de prioridade de cada artefato, a IDE altera a cor de cada um deles. Na Figura 3 é possível perceber como a árvore de elementos fica após a verificação de prioridade dos artefatos. Tornando possível a percepção de quais artefatos são mais essenciais para o negócio.

4. Trabalhos futuros

Existem implementações que podem ser feitas no plugin objetivando a eficiência na identificação de componentes mais importantes. Uma delas seria o envio de requisições com um conjunto de artefatos, atualmente o plugin faz a verificação de uma artefato por vez, isso faz com que fique menos performática a percepção de componentes mais importantes para o negócio.

Outra possível melhoria seria a implementação do plugin em outras plataformas utilizadas pelas desenvolvedoras e desenvolvedores. Como, por exemplo, o VSCode ou Eclipse IDE, que são utilizados também em projetos Java.

Além disso, é possível implementar melhorias para que seja possível a prevenção de dívidas técnicas nos componentes mais importantes para o negócio. Atualmente, não existe nada relacionado à prevenção, existe um alerta para que a pessoa desenvolvedora consiga perceber quais elementos mais importantes para o negócio.

5. Considerações finais

Como uma forma de identificação de quais componentes são mais prioritários para o negócio foi criado um plugin que efetua uma comunicação com o Tracy-TD e informa a pessoa desenvolvedora quais componentes fazem parte do core dos projetos. Essa implementação foi feita com alguns desafios técnicos superados.

O processo de desenvolvimento de plugin se mostra bastante desafiador pois envolve uma série de conceitos e tecnologias que não são comuns na maioria das soluções em softwares existentes. Por isso, é necessário que haja um estudo em documentação e em código existente. Além disso, é importante que a pessoa desenvolvedora saiba com clareza quais recursos e objetivos que o plugin irá utilizar.

6. Referências

ALMEIDA, R. R. de et al. **Aligning Technical Debt Prioritization with Business. A Multiple-Case Study** 2019.

POTDAR, A.; SHIHAB, E. **An exploratory study on self-admitted technical debt**. In: 2014 IEEE International Conference on Software Maintenance and Evolution. [S.l.:s.n.], 2014. p. 91–100. 2

WEHAIBI, S.; SHIHAB, E.; GUERROUJ, L. **Examining the impact of self-admitted**

technical debt on software quality. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). [S.l.: s.n.], 2016. v. 1, p.179 -- 188. 1

GRIFFITH, I. et al. **A simulation study of practical methods for technical debt management in agile software development.** In: Proceedings of the Winter Simulation Conference 2014. [S.l.: s.n.], 2014. p. 1014–1025. 2