Dashboard para suportar tomadas de decisões na gerência de dívidas técnicas - Uma aplicação na Indústria

Ruan Cruz Soares Silva, Rodrigo Rebouças de Almeida

Departamento de Ciências Exatas – Universidade Federal da Paraíba (UFPB)
Rio Tinto – PB – Brazil
[1]{ruan.cruz, rodrigo.rebouças}@dcx.ufpb.br

ABSTRACT. *The development of a dashboard module to support decision-making in technical debt management in software projects, as presented in this study, is of great importance. The module allows for the definition of specific objectives and key results for reducing technical debt, as well as real-time progress tracking. This solution significantly contributes to improving quality and reducing risks in software development projects, crucial factors for the success of any software project. By addressing technical debt, which is often neglected or postponed, the module can help ensure the long-term viability and sustainability of software systems.*

RESUMO. *O desenvolvimento de um módulo de dashboard para suportar tomadas de decisões na gerência de dívidas técnicas em projetos de software é de extrema importância. Esse módulo permite a definição de objetivos e resultados-chave específicos para a redução das dívidas técnicas, além de fornecer um acompanhamento em tempo real do progresso. Essa solução contribui de forma significativa para melhorar a qualidade e reduzir os riscos em projetos de desenvolvimento de software, fatores cruciais para o sucesso de qualquer projeto nessa área. Ao abordar as dívidas técnicas, frequentemente negligenciadas ou adiadas, o módulo auxilia na garantia da viabilidade e sustentabilidade de longo prazo dos sistemas de software.*

1. Introdução

A gestão de dívidas técnicas é um desafio constante para as empresas que desejam manter sua infraestrutura tecnológica atualizada e eficiente. Essas dívidas são compromissos adquiridos durante o processo de desenvolvimento de software que, quando não atendidos, resultam em custos adicionais e impactos negativos na qualidade do software. Reconhecendo a importância desse problema, o presente trabalho de conclusão de curso tem como objetivo desenvolver um dashboard de gerenciamento de dívidas técnicas, fornecendo uma ferramenta eficaz para auxiliar as tomadas de decisões na gestão dessas dívidas.

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.

Com o aumento do uso de tecnologias e plataformas digitais, o volume e a complexidade das dívidas técnicas acumuladas têm se intensificado, afetando negativamente a produtividade e a qualidade dos produtos desenvolvidos. Nesse contexto, a visualização de dados por meio de gráficos desempenha um papel fundamental no gerenciamento das dívidas técnicas e na sustentabilidade dos negócios.

Diante dessa realidade, o trabalho propõe a criação de um dashboard com gráficos que evidenciam a acumulação de dívidas técnicas, fornecendo suporte para a tomada de decisões relacionadas ao gerenciamento dessas dívidas. Esse dashboard faz parte de uma ferramenta chamada Tracy-TD, uma solução de gerenciamento de dívidas técnicas voltada para o contexto empresarial. Por meio do dashboard proposto, os usuários poderão acessar informações relevantes sobre o acúmulo de dívidas técnicas em tempo real, utilizando-as como base para estabelecer objetivos e metas no gerenciamento dessas dívidas.

Conforme definido por Schnaide et al. (2004), é importante ressaltar que os benefícios obtidos por meio da medição decorrem das decisões e ações tomadas a partir da análise dos dados, e não apenas da coleta desses dados. Assim, a visualização dos resultados desempenha um papel crucial na metodologia de estudo das dívidas técnicas. Gráficos e relatórios que apresentam os resultados das análises e métricas podem ajudar a identificar padrões e tendências, bem como a priorizar as decisões técnicas mais críticas para o negócio.

Portanto, este trabalho de conclusão de curso busca contribuir para a área de gerenciamento de dívidas técnicas, oferecendo uma solução prática e eficiente para auxiliar as empresas no gerenciamento dessas dívidas, visando melhorar a produtividade e a qualidade dos produtos desenvolvidos.

2. Metodologia e Cronograma

2.1. Metodologia

A metodologia contemplada para criação do Módulo de dashboard relacionados a dívidas técnicas envolveu os seguintes passos:

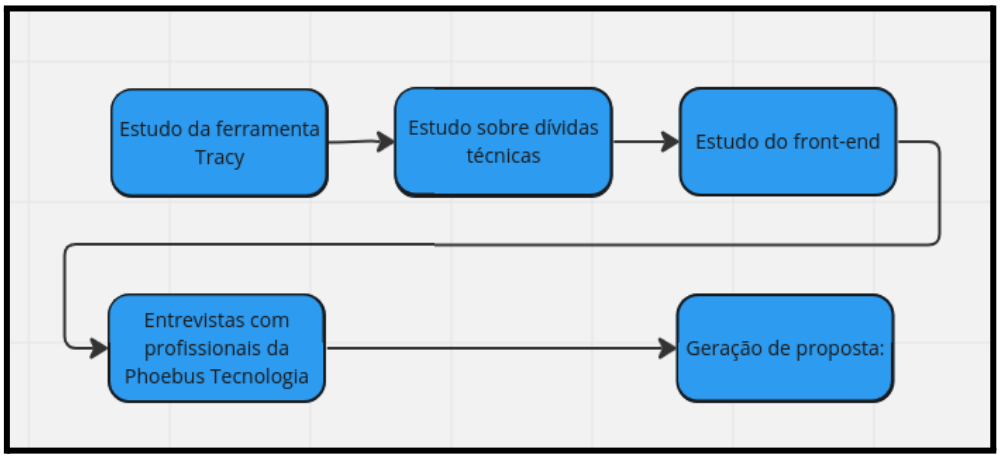


Figura 1 - Fluxograma da metodologia

- Estudo da ferramenta Tracy-TD: O primeiro passo foi entender o funcionamento e as principais características da ferramenta Tracy-TD. Foram realizadas pesquisas bibliográficas e estudos de caso para compreender como a ferramenta era utilizada para gerenciamento de dívidas técnicas.

- Estudo sobre dívidas técnicas: Em seguida, foram realizados estudos sobre dívidas técnicas, suas definições e tipos, bem como sua importância na gestão de projetos de software.

- Entrevistas com profissionais da Phoebus Tecnologia: Foram realizadas entrevistas com três profissionais da Phoebus para identificar os principais gráficos utilizados para acompanhar e gerenciar dívidas técnicas na empresa.

- Estudo do front-end: Estudou-se Angular e seus componentes necessários para geração de gráficos com Apexcharts.

- Geração de proposta: Com base nos estudos e entrevistas realizados, foi gerada uma proposta de dashboard com gráficos de acúmulo de dívidas técnicas para suportar decisões sobre o gerenciamento de dívidas técnicas.

3. Desenvolvimento

3.1 Fundamentação teoria

A gestão eficiente de dívidas técnicas é crucial para o sucesso de um projeto de software. Como afirma Martin Fowler(2009), uma das principais referências na área de desenvolvimento de software, "a dívida técnica é um conceito útil para entender o custo de adiar uma parte do trabalho". Isso significa que, em algum momento, a dívida técnica terá que ser paga, seja em termos de tempo, recursos financeiros ou de qualidade do produto final.

Além disso, é importante ressaltar a relação entre a medição e análise de dados e a gestão de dívidas técnicas. Segundo Boehm (2014)**,** "medidas técnicas de software são necessárias para suportar tomada de decisão de gerenciamento de projeto". Através da análise de dados, é possível identificar os pontos críticos que precisam ser melhorados e, assim, definir estratégias mais eficientes para lidar com as dívidas técnicas.

Conforme salienta o autor Steve McConnell(2014), "dívidas técnicas são como dívidas financeiras. Permitem que você faça algo agora mais rápido, mas isso virá com um custo mais tarde". Por isso, é importante que as empresas tenham um sistema eficiente de gerenciamento de dívidas técnicas, para evitar que as mesmas se acumulem e gerem prejuízos no futuro.

3.2. Problemática e desafios

Dívidas técnicas são um dos grandes desafios enfrentados por empresas de desenvolvimento de software. Elas são definidas como "um conceito metafórico que descreve o custo futuro de manter um código-fonte de má qualidade ou incompleto"

(Cunningham, 1992). A falta de organização no controle dessas dívidas pode levar a atrasos, aumento de custos e perda de qualidade no produto final.

Uma das principais causas de dívidas técnicas é a pressão para entregar projetos rapidamente, muitas vezes sacrificando a qualidade do código ou a adição de funcionalidades adicionais. Como consequência, muitas empresas sofrem pela falta de organização para o controle de dívidas técnicas, o que pode gerar uma acumulação dessas dívidas ao longo do tempo. Essas dívidas acumulam-se ao longo do tempo, tornando o software mais difícil e custoso de ser mantido e atualizado. Alguns dos principais problemas e desafios das dívidas técnicas são:

- Aumento dos custos de manutenção: Dívidas técnicas podem tornar o software mais difícil de ser mantido, aumentando os custos de manutenção.
- Diminuição da qualidade: Dívidas técnicas podem levar a um código fonte de menor qualidade, tornando o software mais propenso a erros e defeitos.
- Atrasos no cronograma: Dívidas técnicas podem atrasar o cronograma de um projeto, pois o tempo é necessário para corrigir os problemas.
- Redução da eficiência: Dívidas técnicas podem reduzir a eficiência do desenvolvimento, uma vez que os desenvolvedores podem ter que passar mais tempo resolvendo problemas em vez de implementando novas funcionalidades.

4. Solução Proposta

4.1 Descrição geral da solução proposta:

Para lidar com esses problemas e desafios, é importante implementar boas práticas de gerenciamento de dívidas técnicas, como a identificação precoce das dívidas, o monitoramento contínuo do código-fonte e a utilização de ferramentas de visualização de dados, e essas práticas podem ajudar a manter as dívidas técnicas sob controle e garantir que o software esteja sempre atualizado com alta qualidade.

Para isso, a ferramenta Tracy, tem um módulo que permitiria a criação de filtros para que as informações do dashboard possam contemplar as mais diversas variáveis que uma dívida técnica possa ter, como a sua classificação como alta, média ou baixa, o produto em que está inserida, as features afetadas, a sua prioridade e os componentes envolvidos.

Com esse dashboard, seria possível visualizar de forma clara e objetiva a evolução das dívidas técnicas ao longo do tempo, identificar quais são as mais críticas urgentes para serem resolvidas e monitorar o progresso das ações que estão sendo tomadas para reduzir essas dívidas.

Dessa forma, o dashboard de gráficos cumulativos por meio da ferramenta Tracy seria uma solução eficaz para lidar com os desafios das dívidas técnicas, proporcionando uma maior organização e controle das informações, auxiliando na tomada de decisão e, consequentemente, melhorando a qualidade dos projetos de software.

Por fim, é fundamental que as empresas tenham uma cultura organizacional voltada para a gestão de dívidas técnicas. Como afirma Martin Fowler (2018), "a chave é manter sua dívida técnica em um nível gerenciável, para que você possa continuar a entregar software com eficiência". Para isso, é necessário um comprometimento da equipe em manter um código limpo e organizado, evitando a acumulação de dívidas técnicas que possam prejudicar o projeto a longo prazo

4.2 Desenvolvimento do módulo de Dashboard

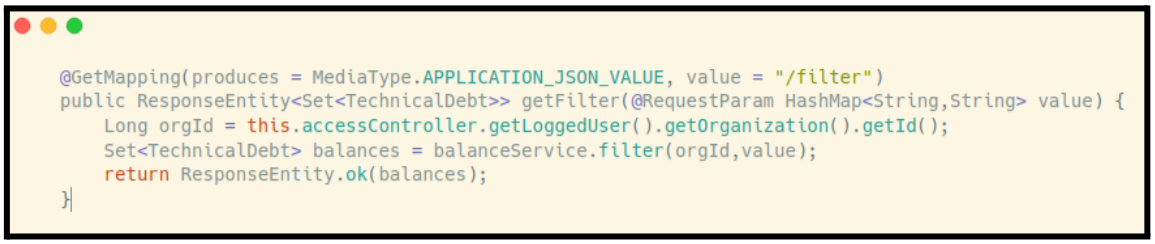
O desenvolvimento de um novo módulo de dashboard foi realizado dentro da estrutura já existente do Tracy. A seguir vamos resumir como foi o desenvolvimento desse módulo e seus principais pontos para evolução do dashboard.

4.2.1 Framework Apexcharts

O Apexcharts é um framework de visualização de dados que oferece uma ampla variedade de opções de gráficos interativos. Ele foi escolhido para gerar e manipular os gráficos exibidos no dashboard, fornecendo uma experiência visual atraente e informativa para os usuários.

4.3 Desenvolvimento do dashboard no Back-end

No desenvolvimento do dashboard a arquitetura do sistema que contempla Java, Spring Boot e PostgreSQL no backend, foram dados continuidades aos princípios do padrão de projeto Model-View-Controller (MVC) que já se encontrava no projeto, para organizar as camadas do sistema e fazer novos serviços em que possam receber as requisições do front-end, manipular, tratar e dar um retorno adequado para o usuário do sistema.



```
@GetMapping(produces = MediaType.APPLICATION_JSON_VALUE, value = "/filter")
public ResponseEntity<Set<TechnicalDebt>> getFilter(@RequestParam HashMap<String,String> value) {
    Long orgId = this.accessController.getLoggedUser().getOrganization().getId();
    Set<TechnicalDebt> balances = balanceService.filter(orgId,value);
    return ResponseEntity.ok(balances);
}
```

Figura 2: Classe de controle do Dashboard

O controlador é responsável por receber as requisições do cliente, processar os dados e retornar as respostas apropriadas. Ele age como intermediário entre a camada de visualização e a camada de modelo, nesse caso da figura 1 ele recebe os filtros que foram passados no front end e repassa para a classe de serviço do Dashboard para identificar qual filtro foi selecionado.

```
public Set<TechnicalDebt> filter(Long organizationId, HashMap<String,String> value) {
        Set<TechnicalDebt> aux = new LinkedHashSet<>();
        for (Map.Entry<String, String> entry : value.entrySet()) {
            String key = entry.getKey();
            Arrays.stream(entry.getValue().split(",")).forEach((vl) -> {
                if (key.equals("technicalPriority")) {
                    switch (vl) {
                        case "0":
                            aux.addAll(this.findHigh(organizationId));
                            break;
                        case "1":
                            aux.addAll(this.findMedium(organizationId));
                            break;
                        case "2":
                            aux.addAll(this.findLow(organizationId));
                            break;
                    }
                }
                if(key.equals("configItem")){
                    var valor = Integer.parseInt(vl);
                    aux.addAll(this.debtRepository.findByComponentsByTechnicalDebt(valor));
                }
                if(key.equals("state")){
                    var valor = Integer.parseInt(vl);
                    aux.addAll(this.debtRepository.findByStatesByTechnicalDebt(valor));
                }
                if(key.equals("product")){
                    var valor = Integer.parseInt(vl);
                    aux.addAll(this.debtRepository.findByProductByTechnicalDebt(valor));
                }
                if(key.equals("feature")){
                    var valor = Integer.parseInt(vl);
                    aux.addAll(this.debtRepository.findByFeatureByTechnicalDebt(valor));
                }
                if(key.equals("businessPriority")){
                    var valor = Integer.parseInt(vl);
                    aux.addAll(this.debtRepository.findByBussinesPriorityByTechnicalDebt(valor));
                }
            } );

        }
        return aux;
    }
```

Figura 3: Classe de serviço do Dashboard

Os Serviços são responsáveis pela lógica de negócio da sua aplicação, além de ser responsável por se comunicar com as camadas mais internas do Software, como por exemplo, uma camada de Dados. Na figura 2, procuramos identificar qual foi o filtro selecionado e o seu identificador (valor) para encaminhar para a busca no banco de dados, vale ressaltar que utilizamos a estrutura de dados LinkedHashSet, que é uma implementação da interface Set no Java que mantém a ordem de inserção dos elementos e não permite elementos duplicados. Ele combina as características do HashSet e do LinkedList, fornecendo a eficiência de busca de um conjunto de hash e a ordenação de uma lista encadeada.

```
 @Query(value = "select distinct on (td.id,td.identification_date) * from technical_debt td " +
                        "join asset a " +
                            "on a.organization_id = td.organization_id " +
                        "join asset_ci ac " +
                            "on ac.ci_id = td.ci_id " +
                        "where ac.asset_id =?1 order by td.identification_date asc" ,nativeQuery =
true)
    List<TechnicalDebt> findByProductByTechnicalDebt(Integer asset_id);

    @Query(value = " select td.* from impact i " +
                        " join technical_debt td" +
                        "   on td.id = i.td_id " +
                        "join business_process bp" +
                        "   on bp.id = i.bp_id " +
                        "where bp.id =?1 and i.status = 'ACTIVE' order by td.identification_date asc; "
,nativeQuery = true)
    List<TechnicalDebt> findByFeatureByTechnicalDebt(Integer id);
```

Figura 4: Classe do Repositório do Dashboard

A classe de repositório é um padrão de projeto similar ao DAO (Data Access Object) no sentido de que seu objetivo é abstrair o acesso a dados de forma genérica a partir do seu modelo. Na figura 3, realizamos a busca das dívidas técnicas de acordo com o filtro selecionado. Utilizando a anotação @Query, podemos construir um script para consultar o banco de dados, ordenando sempre de forma ascendente pelas datas das dívidas técnicas identificadas.

4.4 Desenvolvimento do dashboard no Front-end

O desenvolvimento do dashboard em Angular abrange várias tecnologias e frameworks, como Angular, TypeScript, JavaScript, CSS e HTML. A seguir, descreve cada camada e seus componentes específicos;

4.4.1 Camada de Componentes

No Angular, os componentes são as principais unidades de construção da interface do usuário. Eles são responsáveis por exibir os dados, responder a eventos e gerenciar o estado da interface. Cada componente é composto por um arquivo TypeScript que define a lógica do componente, um arquivo HTML que define a estrutura e os elementos da interface, um arquivo CSS que estiliza o componente e um arquivo de teste opcional.

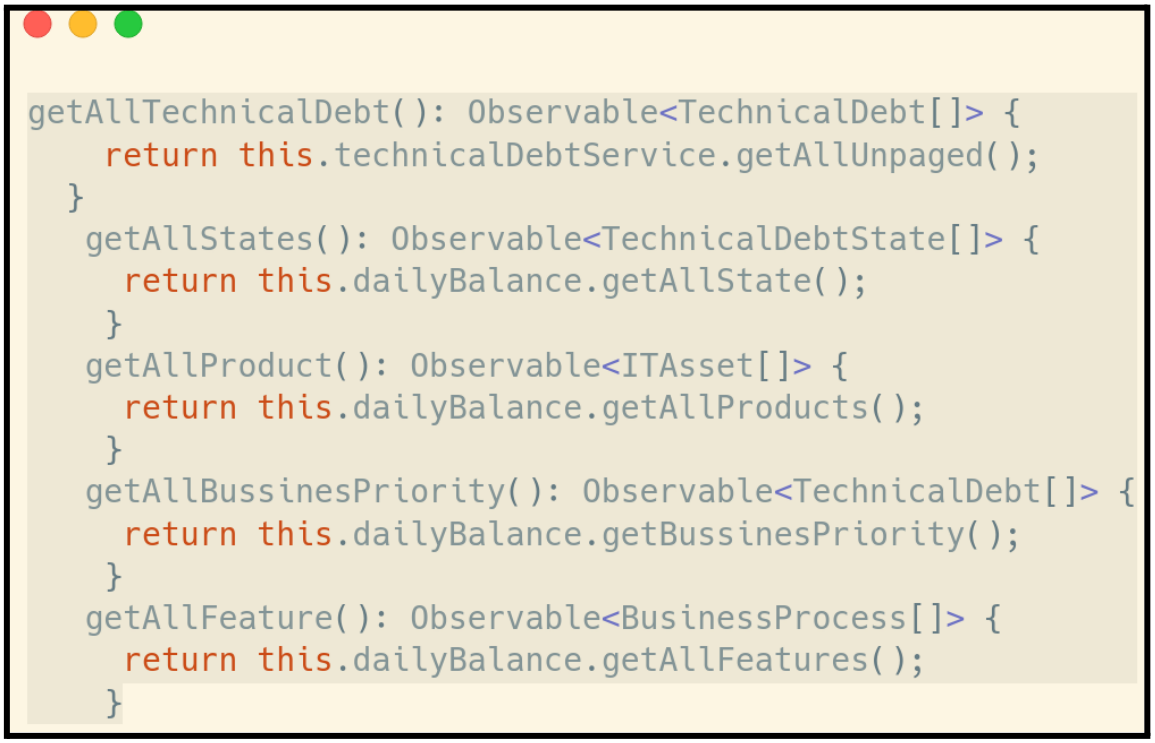
```
@Component({
    selector: 'project',
    templateUrl: './project.component.html',
    encapsulation: ViewEncapsulation.None,
    changeDetection: ChangeDetectionStrategy.OnPush
})
export class ProjectComponent implements OnInit, OnDestroy {
    @ViewChild("chart") chart: ChartComponent;
    @ViewChild(MatPaginator, { static: true }) paginator: MatPaginator;
    @ViewChild('contextMenuTrigger', { static: true }) contextMenuTrigger: MatMenuTrigger;
    @ViewChild('sortButton') sortButton: MatButton;
    @ViewChild('bPriorityFilterButton') bPriorityFilterButton: MatButton;
    @ViewChild('techPriorityFilterButton') techPriorityFilterButton: MatButton;
    @ViewChild('techImpactFilterButton') techImpactFilterButton: MatButton;
    @ViewChild('techEffortFilterButton') techEffortFilterButton: MatButton;
    @ViewChild('typeFilterButton') typeFilterButton: MatButton;
    @ViewChild('configItemFilterButton') configItemFilterButton: MatButton;
    @ViewChild('metricFilterButton') metricFilterButton: MatButton;
    @ViewChild('milestoneFilterButton') milestoneFilterButton: MatButton;
    @ViewChild('issueStatusButton') issueStatusButton: MatButton;
    @ViewChild('featureFilterButton') featureFilterButton: MatButton;
```

Figura 5: Classe project.component.ts do dashboard

Essas classes são chamadas de "classes de componente" e são usadas para definir os componentes da interface do usuário da sua aplicação Angular. A classe de componente possui propriedades e métodos que são usados para controlar o comportamento do componente e fornecer a lógica da aplicação. Na figura 1, podemos ver que o comando @ViewChild é usado para obter uma referência a um elemento específico do template dentro de um componente Angular, permitindo a interação com ele através da propriedade associada.No caso o objetivo é obter uma referência a um elemento do template como por exemplo o identificador 'configItemFilterButton' e armazená-lo na propriedade configItemFilterButton do componente.

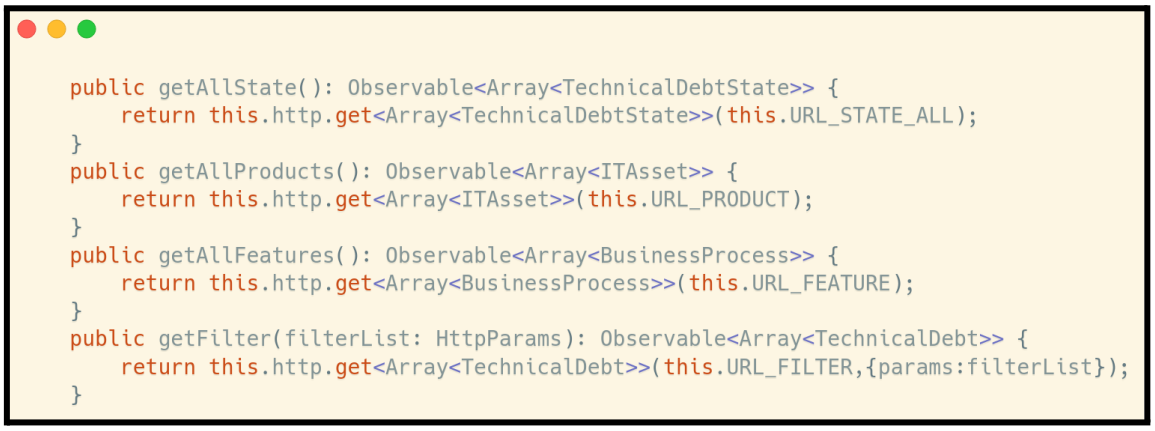
4.4.2 Camada de Serviços

Os serviços no Angular são responsáveis por fornecer funcionalidades e dados compartilhados entre componentes. Eles são usados para buscar dados de APIs, executar lógica de negócios e compartilhar informações entre componentes.



```
getAllTechnicalDebt(): Observable<TechnicalDebt[]> {
    return this.technicalDebtService.getAllUnpaged();
  }
   getAllStates(): Observable<TechnicalDebtState[]> {
     return this.dailyBalance.getAllState();
    }
   getAllProduct(): Observable<ITAsset[]> {
     return this.dailyBalance.getAllProducts();
    }
   getAllBussinesPriority(): Observable<TechnicalDebt[]> {
     return this.dailyBalance.getBussinesPriority();
    }
   getAllFeature(): Observable<BusinessProcess[]> {
     return this.dailyBalance.getAllFeatures();
    }
```

Figura 6: Classe SharedService.ts do dashboard



```
public getAllState(): Observable<Array<TechnicalDebtState>> {
    return this.http.get<Array<TechnicalDebtState>>(this.URL_STATE_ALL);
}
public getAllProducts(): Observable<Array<ITAsset>> {
    return this.http.get<Array<ITAsset>>(this.URL_PRODUCT);
}
public getAllFeatures(): Observable<Array<BusinessProcess>> {
    return this.http.get<Array<BusinessProcess>>(this.URL_FEATURE);
}
public getFilter(filterList: HttpParams): Observable<Array<TechnicalDebt>> {
    return this.http.get<Array<TechnicalDebt>>(this.URL_FILTER,{params:filterList});
}
```

Figura 7: Classe TechDebtDailyBalanceService.ts do dashboard

A classe SharedService da figura 2, é uma classe de serviço que centraliza a lógica e funcionalidades compartilhadas entre vários componentes. Ela pode ser injetada e utilizada por outros componentes ou serviços dentro da aplicação. Pegando como exemplo O método getAllStates() que retorna um Observable que emite um array de objetos TechnicalDebtState, Dentro desse método, é chamado o método getAllState() do serviço TechDebtDailyBalanceService. Esse método é responsável por obter todos os estados de dívida técnica a partir de alguma fonte de dados.

Já a classe TechDebtDailyBalanceService da figura 3, é um serviço no Angular que lida com a comunicação com uma API relacionada ao balanço diário das dívidas técnicas. Em resumo esses métodos fazem requisições para a consumir a API e

consultar as situações atuais das dívidas técnicas, na medida que o usuário for selecionando os filtros

4.4.3 Camada de apresentação

A camada de apresentação no HTML é responsável por definir a estrutura e os elementos visuais da interface do usuário. O HTML (HyperText Markup Language) é a linguagem de marcação usada para criar e organizar os elementos da página web. Nessa camada, são definidos os componentes visuais, como botões, campos de entrada, tabelas, gráficos e outros elementos que compõem a interface do dashboard.

```
  <button mat-button (click)="openFilterModal('businessPriority');" #bPriorityFilterButton>
    Business Priority <mat-icon>arrow_drop_down</mat-icon>
</button>
<button mat-button (click)="openFilterModal('technicalPriority');" #techPriorityFilterButton>
    Tech. Priority<mat-icon>arrow_drop_down</mat-icon>
</button>
<button mat-button (click)="openFilterModal('configItem');" #configItemFilterButton>
    Components<mat-icon>arrow_drop_down</mat-icon>
</button>
<button mat-button (click)="openFilterModal('product');" #productFilterButton>
    Products<mat-icon>arrow_drop_down</mat-icon>
</button>
<button mat-button (click)="openFilterModal('feature');" #featureFilterButton>
    Features <mat-icon>arrow_drop_down</mat-icon>
</button>
```

Figura 8: Classe project.component.html do dashboard

Na figura 4, o HTML fornecido representa alguns botões com alguns rótulos como o de "Products" por exemplo. A tag <button mat-button>: Este é um elemento <button>. Ele representa um botão com estilo predefinido.
Já a função (click)="openFilterModal('product')": Esta é uma diretiva de vinculação de eventos no Angular. Quando o botão é clicado, o evento click é acionado e o método openFilterModal('product') é chamado. O valor 'product' é passado como argumento para o método.

4.5 Desenvolvimento do dashboard Apexcharts:

O Apexcharts é uma biblioteca de gráficos JavaScript de código aberto que permite a criação de gráficos interativos e visualmente atraentes em páginas da web. Ele oferece uma ampla variedade de tipos de gráficos, como linhas, barras, áreas, pizza, radar e muito mais.

Com o Apexcharts, é possível personalizar completamente a aparência dos gráficos, incluindo cores, estilos de linhas, fontes e elementos interativos, como dicas de ferramentas e legendas.

```
<div class="flex flex-col flex-auto">
<apx-chart id="chart"
           class="flex-auto w-full h-80"
           [chart]="chartTD.chart"
           [colors]="chartTD.colors"
           [dataLabels]="chartTD.dataLabels"
           [grid]="chartTD.grid"
           [labels]="chartTD.labels"
           [legend]="chartTD.legend"
           [plotOptions]="chartTD.plotOptions"
           [series]="chartTD.series"
           [states]="chartTD.states"
           [stroke]="chartTD.stroke"
           [tooltip]="chartTD.tooltip"
           [xaxis]="chartTD.xaxis"
           [yaxis]="chartTD.yaxis"></apx-chart>
</div>
```

Figura 9: Código dos estilo do gráfico do dashboard

O comando fornecido configura as opções e estilos de um gráfico usando a biblioteca ApexCharts. configurando elementos gráficos como:

- **fontFamily**: Define a fonte a ser usada no gráfico. Neste caso, está definido como 'inherit', o que significa que a fonte será herdada do elemento pai.
- **enabled**: Define se os rótulos de dados serão exibidos ou não. Neste caso, está definido como true, o que significa que os rótulos de dados serão exibidos.
- **background**: Configura o estilo de fundo dos rótulos de dados. Neste caso, borderWidth está definido como 0, o que significa que não haverá borda no fundo dos rótulos
- **colors**: Define as cores a serem usadas nas séries do gráfico. Neste caso, são definidas três cores diferentes: "#FF0000", "#ff7f50", "#66cdaa". Essas cores serão aplicadas às séries do gráfico.

5. Resultado

O gráfico acumulativo de dívida técnica é uma representação visual que permite acompanhar o acúmulo de dívidas técnicas ao longo do tempo. Ele mostra o total de dívidas técnicas existentes em um projeto de software em determinado período de tempo, considerando todas as suas prioridades (High, Medium ou Low). Esse tipo de gráfico é muito útil para a gestão de projetos de software, pois ajuda a equipe de desenvolvimento a identificar quais áreas estão acumulando mais dívidas técnicas e a definir prioridades para a correção dessas dívidas. Além disso, o gráfico acumulativo pode ser utilizado para monitorar o progresso na redução das dívidas técnicas e para auxiliar na tomada de decisões estratégicas. Em suma, o gráfico acumulativo de dívida técnica é uma ferramenta poderosa para a gestão eficaz de projetos de software.

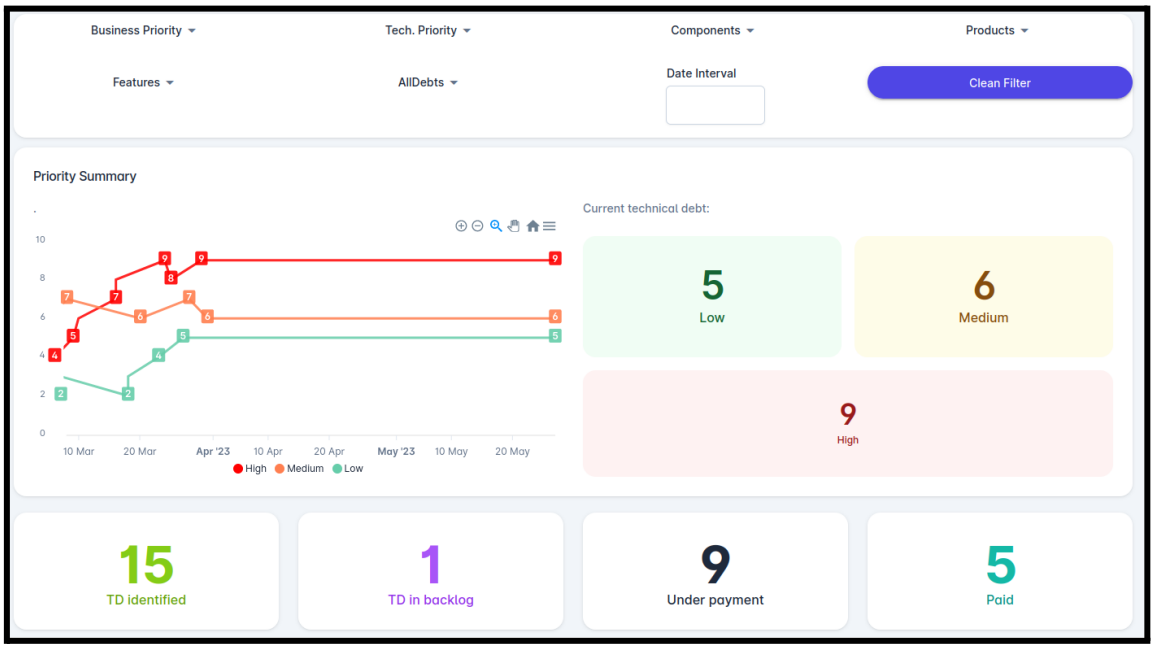


Figura 10: Gráfico Acumulativo

Esse gráfico possibilita uma visão geral das dívidas técnicas presentes no projeto, permitindo que a equipe de desenvolvimento tenha um entendimento claro sobre quais áreas precisam de mais atenção e investimento. Além disso, o gráfico acumulativo permite o monitoramento contínuo do progresso na redução das dívidas técnicas, auxiliando na tomada de decisões estratégicas e priorização de ações para a melhoria da qualidade do software.

O dashboard do Tracy também apresenta informações importantes sobre as dívidas técnicas de um projeto. De acordo com os dados apresentados no gráfico, foram identificadas 16 dívidas técnicas e em backlog existem 0 dívidas. Por outro lado, 9 dívidas técnicas já estão em processo de pagamento, o que significa que as soluções para essas dívidas já estão em andamento. Além disso, o gráfico mostra que 9 dívidas técnicas já foram pagas, ou seja, foram resolvidas de maneira satisfatória. Essas informações são importantes para a gestão do projeto, pois permitem acompanhar o status das dívidas técnicas e definir prioridades para solucioná-las.

Figura 11 - Gráfico com status de dívidas técnicas

O gráfico apresentado no dashboard do Tracy a seguir, mostra a distribuição das dividas técnicas por status de prioridade, indicando que existem 9 dividas técnicas com status Low, 10 com status Medium e 11 com status High. Essa informação é importante para a gestão de projetos de software, pois permite que a equipe de desenvolvimento e os gestores tenham uma visão clara das prioridades e possam tomar decisões estratégicas com base nessas informações. Por exemplo, se houver uma grande quantidade de dívidas técnicas com status High, a equipe pode optar por priorizar o tratamento dessas dívidas em detrimento de outras menos críticas. Dessa forma, o dashboard ajuda a garantir que a equipe esteja focada nas tarefas mais importantes e que o projeto avance de forma eficiente.

Figura 12 - Gráfico com as situações das Dívidas técnicas

Os filtros disponíveis no dashboard para a análise das dívidas técnicas possibilita filtrar as dívidas técnicas por características, como features, produtos, prioridade e componentes, além do status. Ter acesso a essas informações é importante para a tomada de decisão, pois permite identificar quais áreas do sistema estão apresentando mais problemas e quais recursos devem ser alocados para resolvê-los.

Figura 13 - Filtros para os gráficos

Além disso, a identificação dos componentes afetados pelas dívidas técnicas permite que a equipe possa focar seus esforços de correção nos pontos mais críticos do sistema. A partir dessas informações, é possível priorizar a correção das dívidas técnicas de acordo com sua gravidade, evitando que problemas mais críticos comprometam o funcionamento do software. Ter essas informações em mãos é essencial para uma gestão mais eficiente das dívidas técnicas e para garantir a qualidade do software entregue.

6. Conclusão

O desenvolvimento do módulo de dashboard relacionado a dívidas técnicas na ferramenta Tracy-TD se mostrou uma solução prática e eficiente para auxiliar empresas no gerenciamento de suas dívidas técnicas. A importância da gestão de dívidas técnicas se dá pela necessidade de manter a qualidade do software produzido e evitar custos adicionais a longo prazo. Como afirma o autor Steve McConnell em seu livro "Code Complete", "a acumulação de dívida técnica é como acumular dívida financeira. Assim como a dívida financeira acumulada, a dívida técnica acumulada exige juros. Juros na forma de custos adicionais de manutenção, dificuldades de teste, dificuldades de correção de defeitos e dificuldades de evolução" (McConnell, 2004).

O módulo de dashboard desenvolvido na Tracy-TD oferece uma maneira fácil e prática de visualizar as métricas relacionadas às dívidas técnicas e que podem ajudar a traçar metas em OKRs, como proposto por John Doerr em seu livro "Avalie o que Importa". Segundo Doerr, "OKRs" são uma maneira eficaz de estabelecer metas e direcionar a organização para um objetivo comum. Eles têm o poder de alinhar todos os esforços de uma empresa em torno de alguns objetivos-chave" (Doerr, 2018).

O dashboard apresenta gráficos de acúmulo de dívidas técnicas, permitindo uma visão geral do status das dívidas, bem como sua evolução ao longo do tempo. Isso possibilita uma tomada de decisão mais precisa e embasada, como destacado por David J. Anderson e Alexei Zheglov em seu livro "Kanban: Successful Evolutionary Change for Your Technology Business". Segundo os autores, "a tomada de decisão embasada em métricas é um aspecto chave de um sistema de gerenciamento de fluxo, permitindo aos tomadores de decisão uma visão precisa da situação atual e ajudando-os a identificar áreas que precisam de melhoria" (Anderson e Zheglov, 2010).

Em resumo, este trabalho de conclusão de curso teve como objetivo desenvolver uma solução prática e eficiente para o gerenciamento de dívidas técnicas nas empresas por meio da ferramenta Tracy-td. O estudo permitiu entender a importância das dívidas técnicas e como elas podem impactar negativamente no desenvolvimento de projetos de software. Através da entrevista com profissionais da Phoebus Tecnologia, foi possível identificar os filtros mais indicados e gráficos mais relevantes para o acompanhamento e gestão das dívidas técnicas. O módulo de dashboard, desenvolvido com o uso do

framework Angular e do Apexcharts, se mostrou uma ferramenta eficaz para o desenvolvimento do módulo, acompanhamento e tomada de decisão sobre as dívidas técnicas.

Através deste trabalho, esperamos contribuir para que as empresas tenham um norte a seguir no gerenciamento das dívidas técnicas, estabelecendo uma visualização detalhada da atual situação das dívidas técnicas, de forma visual e objetiva por meio dos gráficos. Dessa forma, é possível tomar decisões mais embasadas e reduzir os impactos negativos das dívidas técnicas no desenvolvimento de projetos de software.

Referências

DAGSTUHL SEMINAR. (2016). Managing Technical Debt. Dagstuhl Reports, 6(6), 1-27. doi: 10.4230/DagRep.6.6.1

Doerr, John.(2019). Avalie o que importa: como o Google, Bono Vox e a Fundação Gates sacudiram o mundo com OKRs. Rio de Janeiro: Intrínseca.

SCHNAIDE, L.; BOCHI, E.V.; POZZEBON,(2004). M. Análise de métricas para a gerência de projetos de software. In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 24, Florianópolis.

FOWLER, Martin.(2009). The Technical Debt Quadrant. 14 ago. Disponível em: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html. Acesso em: 31 mar. 2023.

Boehm, B. (2014). "Software Metrics and Models in Software Engineering". In: Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures. Springer, Cham.

CUNNINGHAM, Ward. (1992)The WyCash portfolio management system. In: Proceedings of the National Smalltalk Conference.p. 29-30.

MCCONNELL, Steve.(2004). Code Complete: A Practical Handbook of Software Construction. 2nd ed. Redmond, Wash: Microsoft Press, p. 45.

ANDERSON, David J.; ZHEGLOV, Alexei. (2010). Kanban: Successful Evolutionary Change for Your Technology Business. Seattle: Blue Hole Press.