Rosinaldo Guedes de Lima Junior

Validação de proposta de um modelo para documentação de tomadas de decisão na camada frontend de softwares web

Universidade Federal de Pernambuco
secgrad@cin.ufpe.br
www.cin.ufpe.br

Recife
2023

Rosinaldo Guedes de Lima Junior

Validação de proposta de um modelo para documentação de tomadas de decisão na camada frontend de softwares web

Monografia apresentada ao curso de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

Áreas de Concentração: *Engenharia de Software; Documentação de Software.*
Orientador: *Prof. Vinicius Cardoso Garcia*

Recife
2023

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Junior, Rosinaldo Guedes de Lima.
Validação de proposta de um modelo para documentação de tomadas de decisão na camada frontend de softwares web / Rosinaldo Guedes de Lima Junior. - Recife, 2023.
49 : il., tab.

Orientador(a): Vinicius Cardoso Garcia
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2023.
Inclui referências, apêndices.

1. Documentação de softwares. 2. World Wide Web. 3. Frontend. 4. Desenvolvimento de softwares . 5. Registro de Decisão de Arquitetura (ADR) e Goal Question Metric (GQM). I. Garcia, Vinicius Cardoso. (Orientação). II. Título.

000 CDD (22.ed.)

Dedico este trabalho à minha família, que sempre acreditou em mim, me apoiou e esteve ao meu lado durante a graduação.

AGRADECIMENTOS

À Deus.

À minha mãe, pelo amor incondicional e por apostar em mim até o que não tinha.

À minha irmã, por sempre estar ao meu lado, ser minha amiga e grande inspiração para enfrentar os desafios da vida.

À minha mulher, Andrieli, pelo companheirismo em todos os momentos, pelo amor e carinho que nos une, por toda ajuda e suporte que sempre me deu, mesmo quando a distância.

Ao meu pai, por todo suporte que proporcionou à mim para que pudesse chegar até esse momento.

Ao meu avô, Joca, por todos os aprendizados, por todos os momentos inesquecíveis, por ser minha referência de ser humano.

À todas as pessoas da minha família que acreditaram em mim, me apoiaram e que comemoram minhas conquistas como se fossem suas.

Aos meus colegas de turma, pela parceria nos estudos, pelas risadas e por tornarem a caminhada da universidade tão mais rica. Em especial, Juliano e Valgueiro, pela amizade que extrapolou o ambiente acadêmico.

Ao meu orientador, prof. Vinicius Cardoso Garcia, por todo suporte e ajuda na criação deste trabalho.

A todos que de qualquer forma contribuíram para esta conquista.

Continue a nadar.

– Dory

RESUMO

Nos últimos anos, a demanda por softwares web tem crescido cada vez mais, pois a World Wide Web se tornou um meio vital para comunicação, comércio e entretenimento. Com esse crescimento, aumentaram-se as variedades de finalidades e complexidades de tais softwares, o que gerou a necessidade de enxergá-los, projetá-los e desenvolvê-los de formas distintas, de acordo com suas particularidades. No desenvolvimento web, tais particularidades podem ser facilmente notadas entre os softwares backend e frontend, que são respectivamente a parte do sistema que lida com a lógica de negócios e a interação com o banco de dados e a parte do sistema que lida com a interface e a experiência do usuário. Por esse motivo, assim como nem todo software é igual, consequentemente nem toda documentação de software também será. Além disso, as corridas tecnológicas potencializaram a necessidade de criar softwares de maneira cada vez mais rápida e muitas vezes as decisões tomadas não são registradas ou bem estruturadas, causando, no futuro, dificuldades no controle do desenvolvimento dos programas e dificuldade de adaptação dos possíveis novos colaboradores. Nessa circunstância, a documentação de software tem a finalidade de proporcionar um entendimento completo da construção de uma aplicação e pode ser utilizada como uma ferramenta do próprio processo de escolha em uma tomada de decisão. Por esses motivos, focando nas decisões no escopo da camada de frontend de softwares web, tomando como base documentações de arquitetura de software, mais especificamente modelos de Registros de Decisão de Arquitetura (ADR) e também se inspirando na abordagem de métrica de software Goal Question Metric (GQM), esse trabalho se propôs a criar e investigar um modelo para documentação de tomadas de decisão que fosse estruturado, simples e facilmente compreensível para engenheiros de software. Para isso, inicialmente foram apresentadas as inspirações e motivações do modelo proposto e em seguida o mesmo foi definido, explicado e exemplificado. Depois foi feita uma apresentação para desenvolvedores de software frontend, mostrando os detalhes e as motivações do modelo proposto; e posteriormente aplicou-se um questionário com os mesmos desenvolvedores, para coletar suas opiniões. Os resultados mostraram um bom indício de efetividade de que o modelo de documentação proposto pode registrar decisões de forma estruturada, coesa e simples de criar e compreender, porém houveram algumas sugestões de melhoria. Por fim, este trabalho conseguiu ser efetivo em apresentar um modelo para tomadas de decisão e iniciar as investigações e avaliações sobre o mesmo.

Palavras-chave: Web. Desenvolvimento. Documentação. Métricas. Arquitetura de Software. Front-end. ADR. GQM.

ABSTRACT

In recent years, the demand for web software has been growing increasingly, as the World Wide Web has become a vital medium for communication, commerce, and entertainment. With this growth, the variety of purposes and complexities of such software has increased, which has generated the need to view, design, and develop them in different ways, according to their particularities. In web development, such particularities can be easily noticed between backend and frontend software, which are respectively the part of the system that deals with business logic and interaction with the database and the part of the system that deals with the interface and user experience. For this reason, just as not all software is the same, consequently, not all software documentation will be the same either. Additionally, technological races have amplified the need to create software in an increasingly faster manner, and often decisions made are not registered or well-structured, causing difficulties in controlling the development of programs and difficulty adapting to possible new collaborators in the future. In this circumstance, software documentation aims to provide a complete understanding of the construction of an application and can be used as a tool in the decision-making process. For these reasons, focusing on decisions within the scope of the frontend layer of web software, based on software architecture documentation, more specifically Architecture Decision Records (ADR) models, and also inspired by the Goal Question Metric (GQM) software metric approach, this work aimed to create and investigate a model for decision-making documentation that was structured, simple, and easily understandable for software engineers. To do so, initially, the inspirations and motivations of the proposed model were presented, and then the model was defined, explained, and exemplified. Next, a presentation was made to frontend software developers, showing the details and motivations of the proposed model, and later a questionnaire was applied to the same developers to collect their opinions. The results showed a good indication of the effectiveness of the proposed documentation model in recording decisions in a structured, cohesive, and simple way to create and understand, although there were some suggestions for improvement. Finally, this work was effective in presenting a model for decision-making and initiating investigations and evaluations of it.

Keywords: Web. Development. Documentation. Metrics. Software Architecture. Front-end. ADR. GQM.

LISTA DE FIGURAS

Figura 1 – Tabela de comparação de modelos de captura de decisão de arquitetura. Fonte: [29] 20
Figura 2 – Estrutura do GQM 22

Figura 3 – Diagrama de uso do FDRM 26
Figura 4 – Exemplo de uso do FDRM no Microsoft Word 27
Figura 5 – Exemplo de uso do FDRM em linguagem Markdown: código 28
Figura 6 – Exemplo de uso do FDRM em linguagem Markdown: visualização . . . 29

Figura 7 – Resposta do Dev 1 sobre a decisão que foi solicitada no formulário . . . 37
Figura 8 – Resposta do Dev 3 sobre a decisão que foi solicitada no formulário . . . 37
Figura 9 – Resposta do Dev 4 sobre a decisão que foi solicitada no formulário . . . 38
Figura 10 – Resposta do Dev 6 sobre a decisão que foi solicitada no formulário . . 39

LISTA DE TABELAS

Tabela 1 – **Perfil dos desenvolvedores que responderam o formulário** 30

SUMÁRIO

1 INTRODUÇÃO 12
1.1 CONTEXTO E OBJETIVO 12
1.2 ESTRUTURA DO TRABALHO 13

2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 A WEB 15
2.1.1 Desenvolvimento Frontend 16
2.2 DOCUMENTAÇÃO DE ARQUITETURA DE SOFTWARE 18
2.2.1 Registro de decisão de arquitetura (ADR) 19
2.3 GOAL QUESTION METRIC (GQM) 21
2.4 SUMARIZAÇÃO DO CAPÍTULO 22

3 METODOLOGIA 24
3.1 ESTRATÉGIA PROPOSTA: FDRM 24
3.1.1 Estrutura 25
3.1.2 Aplicação 27
3.1.3 Exemplos 27
3.1.3.1 Exemplo de uso do FDRM no Microsoft Word 27
3.1.3.2 Exemplo de uso do FDRM em linguagem Markdown 28
3.1.4 Validação 29
3.1.4.1 Formulário 30
3.1.4.2 Perfil dos desenvolvedores que responderam o formulário 30
3.2 SUMARIZAÇÃO DO CAPÍTULO 31

4 RESULTADOS E DISCUSSÕES 32
4.1 ANTES DA UTILIZAÇÃO DO MODELO 32
4.1.1 Opiniões 32
4.1.2 Sugestões 33
4.1.3 Interesse em utilizar o modelo 34
4.2 DEPOIS DA UTILIZAÇÃO DO MODELO 34
4.2.1 Opiniões 34
4.2.2 Experiência sobre o uso 35
4.2.3 Respostas da decisão que foi solicitada no formulário 36
4.3 SUMARIZAÇÃO DO CAPÍTULO 40

5 CONSIDERAÇÕES FINAIS 41
5.1 CONCLUSÃO 41

5.2 LIMITAÇÕES DO TRABALHO 42
5.3 TRABALHOS FUTUROS 42

REFERÊNCIAS 43

APÊNDICES 45

A FORMULÁRIO SOBRE O MODELO PROPOSTO DE DOCUMENTAÇÃO DE SOFTWARE FRONTEND 46

1 INTRODUÇÃO

Este capítulo contextualiza os assuntos que motivaram a criação deste trabalho, apresenta o objetivo do mesmo e também descreve sua estrutura.

1.1 CONTEXTO E OBJETIVO

Um software é construído e evolui a partir de um conjunto de decisões que ao longo do seu desenvolvimento estruturam sua arquitetura e suas particularidades de design. E devido à alta variedade de finalidades e complexidades dos softwares demandados atualmente, é importante que os mesmos sejam pensados, especificados, projetados e até mesmo construídos utilizando não apenas tecnologias, mas também metodologias e até estratégias distintas [26].

Além disso, o entendimento completo do histórico da construção de uma aplicação é essencial para uma boa manutenção da mesma no longo prazo [3], pois pode oferecer melhor embasamento para tomada de decisões e portanto, garantir uma melhor gerência da aplicação. Em Lethbridge et al. [14] é mencionado por alguns engenheiros de software, como fator positivo das documentações, que as "informações de arquitetura e outras documentações abstratas geralmente são válidas ou, pelo menos, fornecem orientação histórica que pode ser útil para os mantenedores".

Com a alta competitividade do mercado, "as empresas devem ser capazes de desenvolver sistemas de software de alta qualidade na ‘Velocidade da Internet’" [2], de maneira rápida, e muitas vezes as decisões tomadas não são registradas ou bem estruturadas, causando, no futuro, dificuldades no controle do desenvolvimento dos programas e dificuldade de adaptação dos possíveis novos colaboradores.

Logo, para se ter uma melhor compreensão de um software no longo prazo, se fazem necessárias que as decisões tomadas durante sua criação sejam registradas, bem estruturadas e facilmente entendidas pelos criadores e mantenedores do mesmo. Até porque a única certeza no desenvolvimento de software é a mudança; e quanto maior o sucesso do software, maior o seu ciclo de vida e maior a quantidade de solicitações de mudanças ao longo desse ciclo.

Também é importante mencionar que a documentação de uma decisão em um software pode ser criada depois que a mesma é tomada, mas também pode ser um instrumento que faz

parte do processo de escolha, ajudando a guiar a tomada de decisão, pois ao registrar tal processo enquanto está em andamento, pode-se visualizar com mais clareza os fatores que estão sendo levados em consideração.

Um processo de tomada de decisão pode ser feito com base em fatores quantitativos, qualitativos ou ambos. Tais fatores podem ser coletados através de métricas, que são ideais para fazer comparações e fornecer orientações sobre o que está sendo medido. Vale ressaltar que realizar escolhas através de métricas pode contribuir para evitar que decisões sejam baseadas em achismos ou opiniões enviesadas; e portanto podem torná-las mais assertivas. Como exemplo, em Stark et al. [24] foi dito que as decisões tomadas com base em métricas "ajudaram gerentes e engenheiros a tomar decisões sobre a prontidão do projeto, removendo o otimismo inerente ao 'julgamento de engenharia'".

Apesar dos pontos positivos de documentar os softwares, existem alguns desafios relacionados à criação de documentações. É importante mencionar que muitas vezes os softwares são criados e não são documentados (e consequentemente, não comunicados) de forma efetiva. Além disso, algumas das principais complicações na documentação de softwares incluem: a falta de organização dos documentos; dificuldade para manter a documentação atualizada; dificuldade de acesso aos documentos, que muitas vezes ficam salvos em locais até esquecidos; entre outras complicações.

Nesse contexto, focando nas decisões no escopo da camada de frontend de softwares web e nas particularidades de design de tal camada, ou seja, nos seus detalhes que podem definir como partes específicas do código devem ser desenvolvidas ou até mesmo o que devem utilizar (como por exemplo, o uso de uma biblioteca para renderizar componentes específicos, abordagens de estilização, etc), esse trabalho se propõe a criar e investigar um modelo para documentação de tomadas de decisão que seja estruturado, simples e facilmente compreensível para engenheiros de software, utilizando como base a abordagem de métrica de software GQM e modelos existentes de ADR.

1.2 ESTRUTURA DO TRABALHO

- **Primeiro capítulo** (Introdução): Contextualização dos assuntos que motivaram a criação deste trabalho.
- **Segundo capítulo** (Fundamentação Teórica): Explicação dos conceitos dos temas abordados neste trabalho.
- **Terceiro capítulo** (Metodologia): Explanação do processo de pesquisa deste trabalho.
- **Quarto capítulo** (Resultados e Discussões): Exposição dos resultados obtidos neste trabalho e suas respectivas discussões.

- **Quinto capítulo** (Considerações Finais): Apresentação das conclusões e das limitações deste trabalho.

2

FUNDAMENTAÇÃO TEÓRICA

Este capítulo explana os conceitos dos temas abordados neste trabalho, visando tornar mais clara a compreensão do mesmo.

2.1 A WEB

Nos últimos anos, a World Wide Web (Web) se tornou uma plataforma essencial para a entrega de aplicativos e serviços na Internet. Com bilhões de usuários em todo o mundo, a Web se tornou um meio vital para comunicação, comércio e entretenimento. Além disso, a Web tornou-se um dos principais pilares da economia digital, oferecendo oportunidades sem precedentes para empresas e empreendedores inovarem e crescerem em escala global [5].

Na perspectiva do mercado, são infinitas as possibilidades de software web que podem ser criados e quanto melhor a qualidade do produto, maiores as chances de sucesso. Por isso, atualmente existem muitos fatores que envolvem a criação de uma aplicação web, muito além de apenas codificar e mostrar qualquer conteúdo na tela. É preciso garantir qualidade e isso só pode ser feito quando cada etapa da criação é executada com excelência.

O desenvolvimento de software para a Web é uma das áreas mais importantes e dinâmicas da indústria de tecnologia da informação atual. A Web é um ambiente complexo e em constante evolução, que exige que os desenvolvedores possuam habilidades especializadas em diversas áreas, incluindo design de interface do usuário, desenvolvimento de banco de dados, programação de servidor, segurança, desempenho e muito mais [28] [25].

Uma das principais especializações dentro do desenvolvimento de software para a Web é a divisão entre backend e frontend. O backend refere-se à parte do sistema que lida com a lógica de negócios e a interação com o banco de dados, enquanto o frontend refere-se à parte do sistema que lida com a interface do usuário e a experiência do usuário. Os desenvolvedores de backend geralmente trabalham com linguagens de programação como Python, Ruby, PHP ou Java, enquanto os desenvolvedores de frontend geralmente trabalham com tecnologias como HTML, CSS e JavaScript.

Embora o desenvolvimento de software para a Web possa ser desafiador, também é uma área emocionante e recompensadora, que oferece muitas oportunidades de inovação e

crescimento profissional. Além disso, com a crescente demanda por aplicativos baseados na Web e a constante evolução das tecnologias da Web, os desenvolvedores que possuem habilidades especializadas em backend e frontend estão bem posicionados para ter sucesso na indústria de tecnologia da informação.

Como o objetivo deste trabalho está atrelado ao desenvolvimento frontend, a seguir vamos explicá-lo em mais detalhes.

2.1.1 Desenvolvimento Frontend

O Frontend é a camada de uma aplicação web que é responsável por apresentar a interface gráfica ao usuário final, por meio do navegador web. Ele é responsável por interpretar e renderizar os dados enviados pelo backend para apresentar o conteúdo de forma visualmente atraente e intuitiva. O Frontend não se limita apenas aos recursos visuais, ele também pode apresentar diferentes tipos de mídias, como vídeos e áudios, além de ser responsável por permitir a interação do usuário com a aplicação por meio de botões, links e formulários [1].

O Frontend é uma parte fundamental do desenvolvimento de uma aplicação web, uma vez que é responsável por garantir que a experiência do usuário seja agradável, rápida e intuitiva. Ele exige habilidades técnicas em linguagens de marcação, estilo e programação, como HTML, CSS e JavaScript, além de ser recomendado ter conhecimentos em design de interface do usuário, usabilidade e acessibilidade. Em resumo, o Frontend é responsável por criar a interface visual de uma aplicação web e em conjunto com as equipes de design, garantir que ela seja fácil de usar e atraente para o usuário final [9].

O ecossistema web é demasiadamente formentado por softwares de código aberto, que são grandes impulsionadores de inovações e ferramentas muito difundidas no desenvolvimento dos softwares frontend, como por exemplo, React[1], Angular[2] e Vue JS[3], ferramentas utilizadas para criação de interfaces visuais. Além disso, existem muitas técnicas e programas que oferecem recursos adicionais aos providos pelas ferramentas tradicionais de desenvolvimento, que facilitam a vida dos desenvolvedores, como por exemplo, bibliotecas CSS-in-JS e pré-processadores CSS, que são utilizadas para estilização de componentes frontend.

Por terem sido mencionadas neste trabalho em explicações e exemplos, e também por terem sido utilizadas na metodologia deste trabalho, segue abaixo uma breve descrição de algumas ferramentas utilizadas no desenvolvimento frontend, para proporcionar um conhecimento básico prévio sobre as mesmas:

- JavaScript: É dito em mozilla.org [15] que o "JavaScript (JS) é uma linguagem leve, interpretada e baseada em objetos com funções de primeira classe". É comumente usada na web mas também é empregada em vários outros ambientes sem browser,

[1]Documentação: https://react.dev/
[2]Documentação: https://angular.io/
[3]Documentação: https://vuejs.org/

como por exemplo, node.js. Além disso, o JS é uma linguagem dinâmica que é baseada em protótipos e também suporta múltiplos paradigmas de programação: orientação a objetos, imperativos e declarativos [15].

- CSS: CSS[4] é uma linguagem utilizada para estilizar elementos criados em linguagens de marcação (HTML, SVG, XML, etc), ou seja, tem o propósito de especificar como documentos devem ser apresentados na interface do usuário.

- React: De acordo com a própria documentação, o React é uma biblioteca JavaScript para construção de interfaces de usuário [22]. Foi criada pelo Facebook em 2011, porém atualmente é uma biblioteca de código aberto, mantida pela própria empresa que o criou e também por outros desenvolvedores da comunidade web. Segundo o Stack Overflow, foi o framework web mais utilizado de 2021 [19] e foi em 2022 umas das tecnologias da web mais comuns usadas por desenvolvedores profissionais e aqueles que estão aprendendo a codificar [20].

- Chart.js: Chart.js[5] é biblioteca JavaScript de código aberto que foi criada em 2013 e que tem como propósito a criação de gráficos JavaScript simples e flexíveis, para a web moderna [6]. Se destaca por possuir modelos de gráficos predefinidos e fáceis de usar.

- D3.js: D3.js[6] é uma biblioteca JavaScript de código aberto, criada para manipulação de documentos com base em dados, oferecendo todos os recursos dos navegadores modernos, combinando componentes de visualização poderosos e uma abordagem orientada a dados para manipulação de DOM[7] [7]. Possui grande flexibilidade para criar e personalizar gráficos complexos.

- CSS-in-JS: CSS-in-JS é uma ferramenta que permite usar JavaScript para descrever estilos de componentes frontend, podendo ser utilizada tanto no navegador quanto no lado servidor. Tem sido muito utilizada no desenvolvimento web pois utiliza a flexibilidade do JavaScript para aprimorar as formas de estilizar componentes e proporciona benefícios como remoção de código duplicado, isolamento de estilos, entre outros. Uma das bibliotecas mais famosas que usa o CSS-in-JS é a styled-components[8].

- Pré-processador CSS: Um pré-processador CSS é um interpretador que recebe um código gerado em uma linguagem de estilização específica e converte em CSS. Tais linguagens de estilização geralmente contém funcionalidades adicionais ao CSS, ou

[4]Cascading Style Sheets.
[5]Documentação: https://www.chartjs.org/
[6]Documentação: https://d3js.org/
[7]Document Object Model.
[8]Documentação: https://styled-components.com/

seja, possuem abstrações que facilitam e enriquecem o desenvolvimento, como por exemplo, criação de variáveis, condicionais, funções, entre outras. Além disso, um pré-processador CSS pode minificar os arquivos de estilização de uma aplicação.

Após entendermos a importância do Frontend no desenvolvimento de software para a Web, é necessário abordar a importância da documentação da arquitetura de software. Como mencionado anteriormente, a Web é um ambiente complexo e em constante evolução, que exige que os desenvolvedores possuam habilidades especializadas em diversas áreas. Para garantir que o software permaneça escalável, sustentável e adaptável às mudanças, é fundamental que as decisões tomadas durante o processo de desenvolvimento sejam documentadas de forma clara e estruturada. Com a documentação adequada, os desenvolvedores e mantenedores do software poderão entender e atualizar facilmente a arquitetura do sistema, permitindo que ele continue a atender às necessidades do negócio e dos usuários finais.

2.2 DOCUMENTAÇÃO DE ARQUITETURA DE SOFTWARE

Parnas [21] afirma que "um documento é uma descrição escrita que tem um status ou autoridade oficial e pode ser usada como evidência"; e espera-se que seu conteúdo não contenha ambiguidades ou erros.

Colocando de uma maneira abstrata, a ação de documentar é retratar algo em um instante e sobre um instante. É criar um artefato que pode perdurar muito tempo e ser utilizado como referência para melhor compreensão daquilo que foi evidenciado. Indo além, a ato de documentar não está restrito à escrita, pois também pode criado em formatos de áudio e imagem, por exemplo.

Antes de falar sobre documentar arquitetura de software, é preciso definir tal objeto de registro. Segundo o livro *Software Architecture in Practice* [13], "a arquitetura de software de um sistema é o conjunto de estruturas necessárias para raciocinar sobre o sistema. Essas estruturas compreendem elementos de software, relações entre eles e propriedades de ambos". Existem outras definições de arquitetura de software, entretanto a maioria converge em falar sobre um conjunto de componentes e suas responsabilidades e relações.

Em linhas gerais, documentar uma arquitetura de software é registrar, para cada etapa de sua criação, suas características e regras, seu funcionamento e suas razões e motivações para decisões específicas, para dessa maneira possibilitar que os futuros mantenedores de um software consigam compreender com mais clareza os "porquês do software". E vale salientar que como existem diferentes tipos de software, que apresentam diferentes propósitos, estruturas, entre outras características distintas, é sensato considerar que possam existir diferentes formas de documentá-los.

Agora que sabemos o que é a documentação da arquitetura de software, é importante destacar que existem diversas técnicas e tecnologias disponíveis para facilitar esse processo. Com o crescente interesse em desenvolvimento ágil, muitas ferramentas foram desenvolvidas para permitir que os desenvolvedores documentem a arquitetura de software de forma rápida e

eficiente. Essas ferramentas variam desde diagramas de arquitetura até a geração automática de documentação a partir do código-fonte. A seguir, exploraremos uma solução para documentação de arquitetura de software que ajuda os desenvolvedores a realizem escolhas mais claras e assertivas para o seu projeto.

2.2.1 Registro de decisão de arquitetura (ADR)

Segundo cosmos.network [23], uma decisão arquitetônica é "uma escolha de projeto de software que aborda um requisito funcional ou não funcional que é arquitetonicamente significativo" e os documentos que registram este tipo de decisão são chamados de ADR.

O projeto *Architecture decision record* [10] define um ADR como "um documento que captura uma importante decisão de arquitetura feita junto com seu contexto e consequências". Além disso, Kopp et al. [12] afirma que tais registros "respondem à perguntas do tipo ‘por que’ sobre projetos e tornam explícito o conhecimento tácito", ou seja, o conhecimento relacionado às experiêncas (ou bagagem profissional) dos envolvidos na criação do documento.

Existem diferentes modelos para documentar decisões de arquitetura, que se distinguem na forma como estruturam os elementos que consideram importantes para registrar uma decisão. Tais modelos têm sido criados há vários anos e evoluíram não necessariamente no mesmo passo, mas em conjunto com a evolução das arquiteturas de software. Alguns modelos de captura de decisão de arquitetura são apresentados no trabalho de Zimmermann et al. [29], que contém uma tabela comparativa destacando suas diferentes estruturas. Segue abaixo uma imagem da tabela:

TABLE I. COMPARISON OF PRACTITIONER TEMPLATES FOR ARCHITECTURAL DECISION (AD) CAPTURING

AD aspect (attribute)	IEEE 42010 (Template V2.2)	IBM UMF AD Table	Tyree/ Akerman	Bredemeyer Key Decisions	Nygard ADRs	arc42 Hruschka/Starke	Y-Statements (ABB, [24])
ID	Unique Identifier	ID	(in D-Header)	/	(part of name)	(Section #)	(Id)
Outcome	Statement of the decision	Decision (Made)	Decision	Approach	Decision	Decision	we decided for
Requirements trace (FRs, NFRs)	Correspondence or linkage to concerns	(Derived requirements)	Related requirements	Business drivers, technical drivers	/	/	/
Accountability (Role, Person)	Owner of the decision	/	/	/	/	/	/
Software architecture viewpoint trace	Correspondence or linkage to elements	/	Related artifacts	/	/	/	In the context of
Why-answers	Rationale (linked entity)	Justification	Argument	Conclusion	/	(Question under Decision)	(optional "because" half sentence)
Decision drivers	Forces, constraints	/	(Constraints)	Benefits, Drawbacks	Context	Constraints	facing
Assumptions	Assumptions	Assumptions	Assumptions	/	/	Assumptions	/
Options	Considered Alternatives	Alternatives	Positions	/	/	Considered Alternatives	and neglected
Problem	/	Issue or Problem	Issue	/	/	Problem	/
Decision dependencies	(not in template, but in standard)	Related decisions	Related decisions	/	/	/	/
Categorization, classification	/	Subject Area, Topic	Group(ing)	/	/	(Decision Topic)	/
Name	/	Name	(in D-Header)	<<key decision>>	Title	(Section heading)	/
State of AD making	not in template, but in standard	(not in published example)	Status	/	Status	/	/
Impact	not in template, but in standard	Implications	Implications	Issues/ Considerations	Consequences	/	to achieve, accepting that
Other entries	Timestamps, Citations	Motivation	Notes, Related principles	Notes, Drivers realized	/	/	/
Element count	*9 (template), 11 (standard)*	*13*	*14*	*9 (plus 1-2 in header)*	*5*	*5 (with 14 questions)*	*6*
Scoping help (which ADs to capture?)	Yes	(not in table template, not published)	(anecdotal In article)	2001 white paper	/	(ASRs mentioned)	/
Size (page or word limit) or other hints	/	not published	(example is half a page, table form)	/	1-2 pages per ADR	to be ordered by importance	1 (long) sentence per Y-statement
Publication year	*2011*	*1998 (internal)*	*2005*	*2005*	*2011*	*2012*	*2012*

Figura 1: Tabela de comparação de modelos de captura de decisão de arquitetura. Fonte: [29]

Analisando, por exemplo, os modelos *Nygard ADRs* e *Y-Statements*, podemos notar que apresentam as estruturas mais simples com relação a quantidade de atributos e que ao mesmo tempo resultam em documentos com tamanhos totalmente diferentes, 1-2 páginas e 1 sentença longa, respectivamente.

Apesar de todos os modelos presentes na Figura 1 terem o objetivo comum de documentar uma decisão arquitetural, fica evidente, por causa de suas diferentes estruturas, que não existe uma maneira unânime no que se refere a capturar tal decisão, mas sim múltiplas abordagens que podem se adequar ou não a um contexto específico de uso, ou seja, cabe aos criadores do documento decidir qual modelo melhor se adequa aos seus propósitos.

Apesar do nome ADR fazer menção direta à decisões de arquitetura, é valido questionar se os modelos até aqui descritos podem ser utilizados para registar apenas esse tipo de decisão ou se podem se aplicar à outros tipos de escolha.

No texto *From Architectural Decisions to Design Decisions* [18], Olaf Zimmermann faz uma análise progressiva de decisões arquitetonicamente significativas para decisões de design, ou seja, de decisões que tem alto impacto na arquitetura de um software para decisões com baixo impacto. E como continuação, no texto *ADR = Any Decision Record?* [17], o mesmo autor aborda o termo ADR com o sentido ressignificado de *Architectural Decision Records* para *Any*

Decision Record, apresentando a perspectiva de utilizar modelos de ADR para registrar escolhas mais gerais de design de software e expandindo, portanto, as fronteiras de utilização de tais modelos.

Nesse contexto, na prática, os modelos de ADR podem ser utilizados para registrar qualquer tipo de decisão. E por isso, as comunidades acadêmica e web tem criado modelos para registros de decisões que podem ser utilizados para documentar qualquer decisão considerada relevante durante o desenvolvimento de um código-fonte.

Modelos de ADR costumam ser objetivos, simples de compreender e fáceis de elaborar. Podem ser criados em qualquer formato de texto, porém são comumente feitos em arquivos de linguagem Markdown. Este trabalho foi fortemente inspirado por alguns modelos de ADR existentes que podem ser encontrados no projeto *Architecture decision record* [10].

Por fim, vale ressaltar que apesar das diversas estruturas de ADR serem eficazes para registrar tomadas de decisão em um projeto, é preciso que as mesmas sejam usadas de forma lógica, baseada em evidências, para que tornem-se documentos com argumentos que justifiquem coerentemente a decisão tomada. Para isso, é possível utilizar artifícios para tornar as decisões mais racionais, como por exemplo, aplicar métricas para analisar de forma analítica os elementos envolvidos nas tomadas de decisão.

2.3 GOAL QUESTION METRIC (GQM)

GQM[9] é uma abordagem de métrica de software que estabelece um "sistema de medição direcionado à metas para o desenvolvimento de software"[8]. Ela é baseada na lógica de que é preciso definir metas (objetivos) e perguntas relacionadas aos seus respectivos contextos, para identificar métricas direcionadoras para criação de dados que ajudem a proporcionar respostas sobre tais metas [4, 27]. A aplicação do GQM resulta em um modelo de medição estruturado em três níveis:

- Nível conceitual (Objetivo): Definição dos objetivos a serem alcançados.
- Nível operacional (Pergunta): Conjunto de perguntas utilizado para avaliar e caracterizar os objetivos do nível conceitual.
- Nível quantitativa (Métrica): Conjunto de métricas para responder às perguntas do nível operacional e gerar dados para analisá-las de forma quantitativa.

[9] Tradução literal: Objetivo Pergunta Métrica.

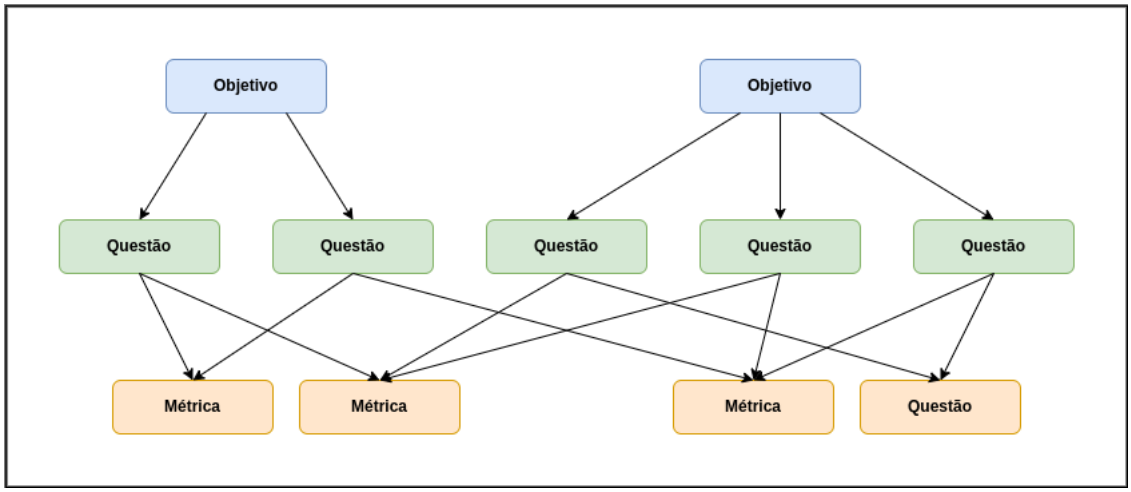


Figura 2: Estrutura do GQM

Na Figura 2, que apresenta um diagrama de exemplo do GQM, podemos notar de forma mais clara como os três níveis da abordagem se relacionam. Inicialmente é preciso definir um objetivo, em seguida são criadas perguntas sobre os objetivos e depois são criadas métricas para responder às perguntas.

Visto que é baseado em métricas e coleta dados focados em um contexto específico, o GQM pode ser utilizado tanto no meio acadêmico quanto no ambiente organizacional, servindo como ferramenta auxiliadora no ciclo de vida produtos, processos e recursos [8].

Em contrapartida, podemos ressaltar que as métricas precisam ser definidas com um alto nível de detalhes para garantir confiabilidade e vale pontuar que o modelo não se preocupa com os problemas relacionados com a medição em si, como viabilidade, economicidade, escalas das medidas e responsabilidade pela análise dos resultados [8].

Por fim, de maneira geral, por proporcionar uma forma organizada e analítica de conduzir a obtenção de dados para alcançar metas, o GQM é uma excelente ferramenta que pode ser utilizada para auxiliar o processo de tomadas de decisão no contexto do desenvolvimento de um software; e portanto também pode contribuir para documentações, visto que o registro de um processo de tomada de decisão pode ser feito enquanto o mesmo está em andamento. Dessa forma, é possível se basear na estrutura do GQM para criar um modelo de documentação que ajude a tomar decisões com mais assertividade.

2.4 SUMARIZAÇÃO DO CAPÍTULO

Este capítulo discorreu sobre todos os assuntos necessários para o entendimento do contexto e das motivações do modelo de ADR criado neste trabalho. Inialmente foi falado sobre a Web e o Desenvolvimento Frontend, pois o modelo criado foca em softwares frontend e é validado com desenvolvedores frontend. Em seguida, falou-se sobre Documentação de Arquitetura de Software e explanou-se sobre ADR, pois o modelo proposto foi baseado em

modelos de ADR já existentes. Por fim, falou-se sobre a abordagem GQM, pois o modelo criado a teve como base para utilizar métricas.

Concluindo, este trabalho referenciou-se nos assuntos abordados neste capítulo, para no contexto do desenvolvimento frontend, fazer uma combinação entre modelos de ADR e a abordagem GQM, visando criar um modelo para registro de tomadas de decisão em softwares frontend. No próximo capítulo serão explanados os detalhes do modelo de ADR proposto.

3

METODOLOGIA

Este capítulo detalha o processo de pesquisa deste trabalho, tornando claras suas inspirações, a natureza, o público, o propósito, a abordagem e os detalhes sobre a estratégia de documentação proposta. Seguem abaixo as informações gerais sobre a metodologia:

- Natureza e Público: Este é um trabalho de natureza *aplicada*, pois tem como um dos seus objetivos implícitos a melhora do processo de documentação de softwares frontend e é focado em desenvolvedores de software frontend que atuam na codificação do código fonte.
- Propósito: Este trabalho tem finalidade *exploratória*, pois propõe-se a criar e investigar uma estratégia de documentação específica, que é originada a partir da combinação de duas ferramentas existentes.
- Abordagem: Este trabalho segue uma abordagem *qualitativa*, pois busca verificar opiniões de desenvolvedores de software frontend quanto à estratégia de documentação proposta.

3.1 ESTRATÉGIA PROPOSTA: FDRM

O modelo FDRM, sigla em inglês para *Frontend Decision Recording based on Metrics* e que se traduz em português como *Registro de Decisão de Frontend baseado em Métricas*, foi criado pelo autor que vos escreve e é uma estratégia de documentação de tomadas de decisão de design de software frontend, baseada em ADR e GQM, que objetiva possibilitar um processo de registro de decisões estruturado, coeso e simples de criar e compreender, baseado em métricas.

- Estruturado porque é dividido em etapas pré-definidas e sequenciais.
- Coeso porque as etapas são complementares e a decisão final é fortemente baseada em métricas. O fluxo de ações até a tomada de decisão é sequencialmente pensado para que a escolha final seja lógica.

- Simples de criar e compreender pois o formato passo a passo torna claro o progresso durante o fluxo das etapas e porque a conexão lógica entre cada passo facilita o entendimento do porquê das informações.

3.1.1 Estrutura

A maioria dos modelos de ADR contém algumas etapas em comum, sendo as duas mais frequentes a etapa que apresenta o contexto (ou descrição do problema) ao qual a decisão está relacionada e a etapa da decisão final. Como exemplo, podemos ver essas etapas presentes no modelo [16] de Michael Nygard.

As etapas intermediárias constumam variar entre cada modelo de ADR, pois são onde geralmente tais modelos se diferenciam pelas suas características específicas.

O modelo FDRM é estruturado em seis (6) etapas distintas e complementares, que devem ser executadas sequencialmente de acordo com a respectiva ordem dos tópicos abaixo.

- Etapa 1 (Contexto): Descreve as circunstâncias relacionadas à criação do documento.
- Etapa 2 (Objetivo): Descreve a decisão que se desejar tomar.
- Etapa 3 (Opções): Lista as opções de escolha disponíveis, de acordo com o objetivo da Etapa 2.
- Etapa 4 (Atributos): Lista características relacionadas às opções da Etapa 3, que serão utilizadas para encontrar contrastes entre tais opções existentes.

  É recomendado adicionar uma breve descrição (de uma frase) do que é cada atributo, para evitar ambiguidades na interpretação de quem for ler a decisão.

  Todo o trabalho foi realizado antes da recomendação acima existir, e portanto o diagrama de uso e as aplicações do modelo FDRM não contém tal recomendação na prática.
- Etapa 5 (Métricas): Classifica o quanto cada atributo da Etapa 4 se aplica a cada opção da Etapa 3, ou seja, todas as opções existentes devem ser avaliadas sob a perspectiva de todos os atributos.

  As classificações (métricas) possíveis são *Bom, Médio e Ruim* e elas devem ser definidas em consenso pela equipe que for utilizar o modelo.

  Existe flexibilidade para definir as classificações porque a equipe que for utilizar o modelo pode detalhar o que significa cada classificação da forma que melhor se adéque ao contexto do projeto ou da decisão.

  Visando manter a coesão do modelo e tornar claro todo o processo que levou à decisão final, deve-se adicionar justificativas para cada métrica aplicada, que também podem incluir links e imagens, por exemplo.

- Etapa 6 (Decisão): Escolhe uma das opções presentes na Etapa 3, com base nas métricas obtidas na Etapa 5.

Sobre as referências (inspirações) para as etapas do modelo, como mencionado anteriormente, as Etapas 1 e 6 são recorrentes nos demais modelos de ADR. As Etapas 2, 4 e 5 são respectivamente uma correspodência direta às etapas do GQM, Objetivos (Goals), Questões (Questions) e Métricas (Metrics). Por fim, a Etapa 2 também está presente em alguns modelos de ADR existentes, como por exemplo no modelo MADR[1] [11].

Para facilitar a compreensão da estrutura do modelo e visualizar como cada etapa se conecta, segue abaixo uma figura que contém o diagrama de uso do modelo FDRM:

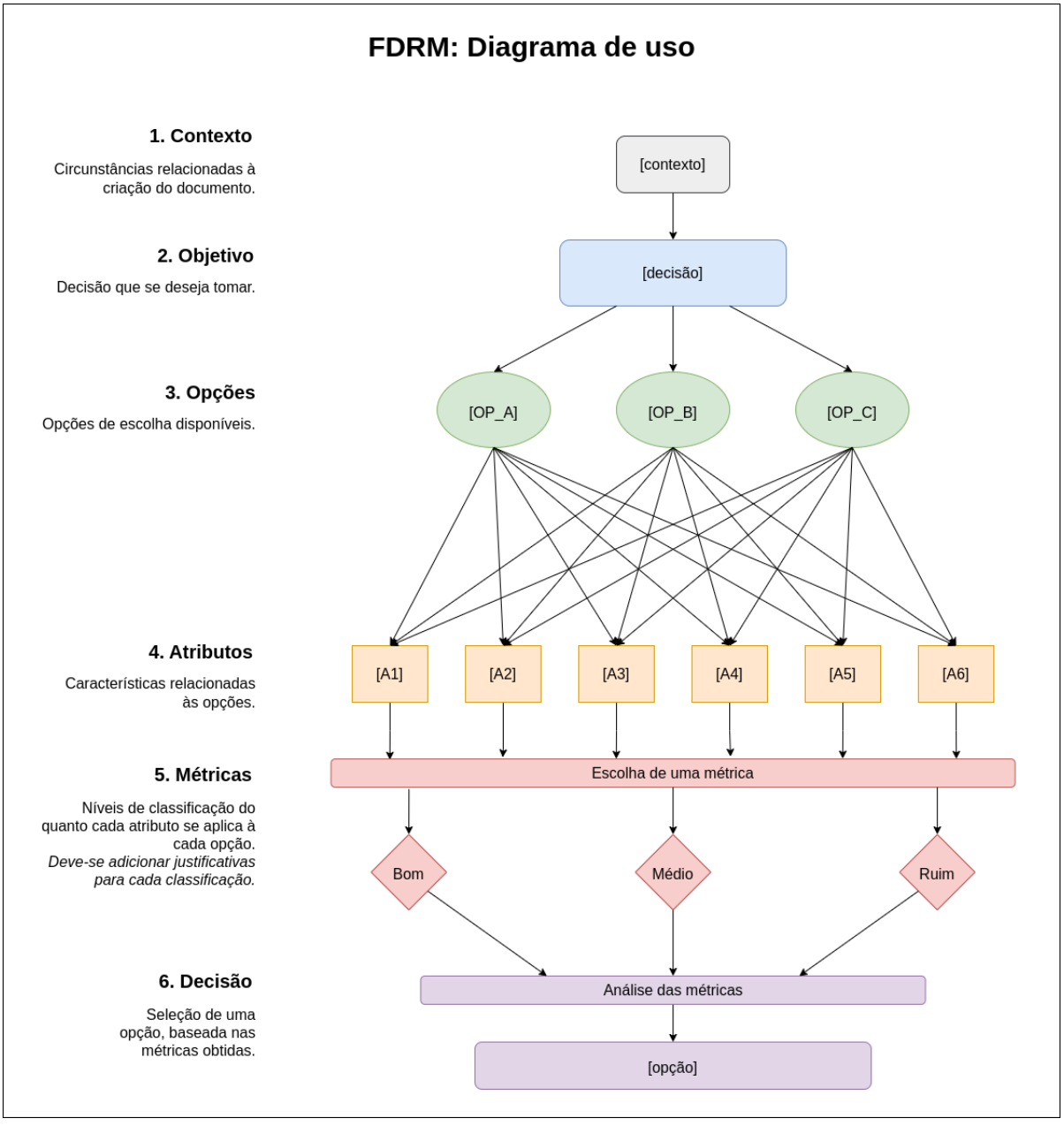


Figura 3: Diagrama de uso do FDRM

[1]MADR: Markdown Any Decision Records

3.1.2 Aplicação

Assim como alguns modelos de ADR presentes em [10], o FDRM foi pensado para ser utilizado dentro do repositório do software que está sendo desenvolvido, em uma pasta dedicada para registrar as decisões tomadas no programa, para que tais escolhas sejam versionadas de acordo com o progresso do software e também para que os registros sejam facilmente acessíveis aos desenvolvedores.

O modelo FDRM pode ser criado em qualquer tipo de arquivo desejado, no entanto, é recomendado que seja construído em linguagem Markdown, por ser uma linguagem simples que é comumemente utilizada pela comunidade de software.

3.1.3 Exemplos

Os dois exemplos abaixo contém o mesmo conteúdo, porém foram feitos em formatos de arquivo diferentes, para demonstrar a flexilidade de uso da ferramenta proposta.

3.1.3.1 Exemplo de uso do FDRM no Microsoft Word

Contexto: Um projeto frontend utilizando o framework React e que precisa escolher uma biblioteca para mostrar gráficos simples de barra e de área. Esta necessidade surgiu a partir dos requisitos do ticket #1010.

Objetivo: Escolher biblioteca para criação de gráficos.

Opções: D3.js (D3), Chart.js (Chart).

Atributos: Curva de aprendizado, personalização e performance.

Métricas

Justificativas

- Curva de aprendizado
  - D3: a criação dos gráficos é feita de uma maneira mais manual.
  - Chart: possui modelos de gráficos predefinidos e simples de usar.
- Personalização
  - D3: possui grande flexibilidade para criar e personalizar gráficos mais complexos.
  - Chart: possui pouca flexibilidade para personalizar os gráficos predefinidos.
- Performance
  - Por ser uma biblioteca mais simples, o Chart é mais leve e rápido em relação ao D3.

Atributos	D3.js	Chart.js
Curva de aprendizado	Média	Boa
Personalização	Boa	Média
Performance	Média	Boa

Decisão: **Chart.js**

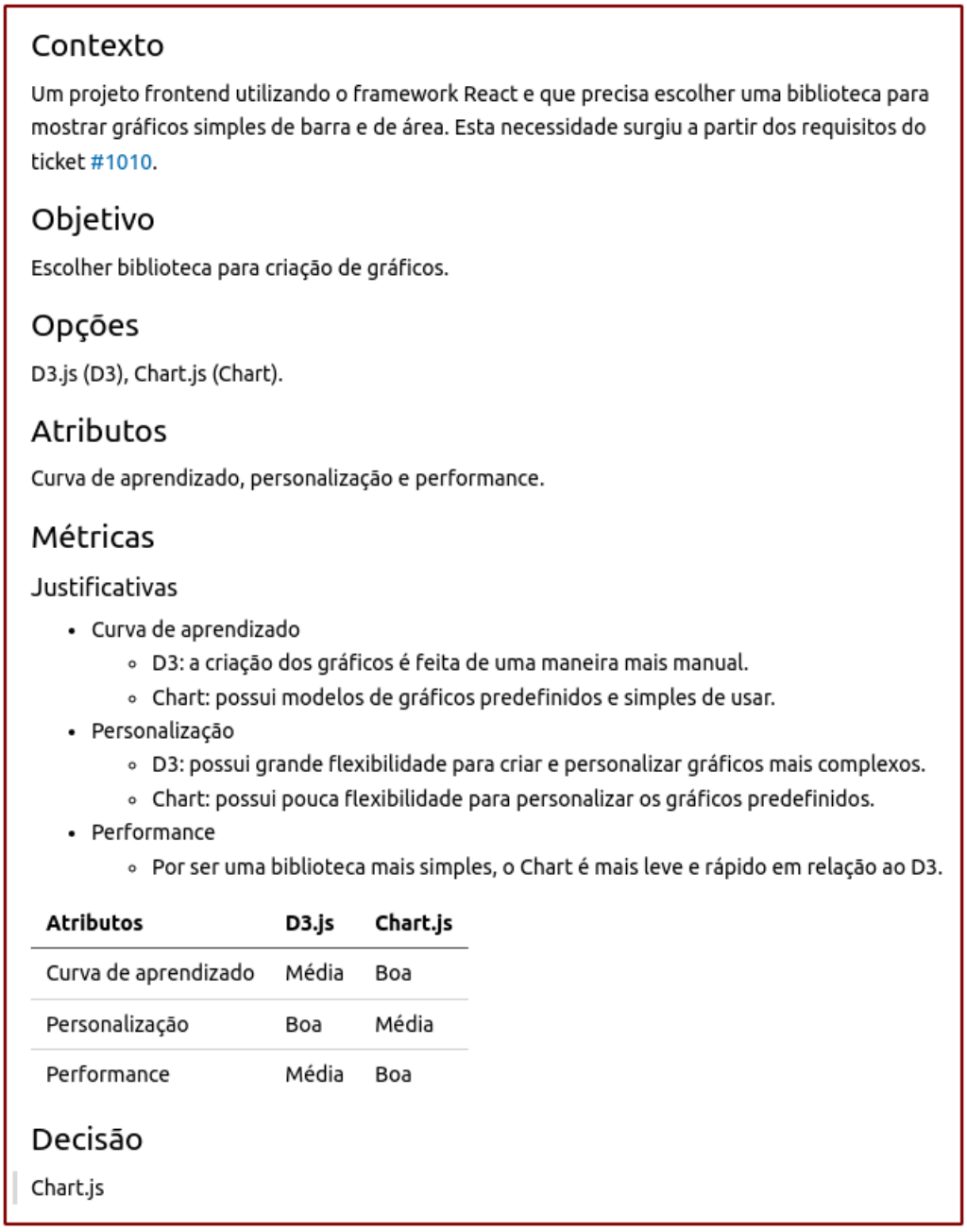
Figura 4: Exemplo de uso do FDRM no Microsoft Word

3.1.3.2 Exemplo de uso do FDRM em linguagem Markdown

decisao_01.md > ## Decisão

```
## Contexto

Um projeto frontend utilizando o framework React e que precisa escolher uma
biblioteca para mostrar gráficos simples de barra e de área. Esta necessidade
surgiu a partir dos requisitos do ticket [     ]().

## Objetivo

Escolher biblioteca para criação de gráficos.

## Opções

D3.js (D3), Chart.js (Chart).

## Atributos

Curva de aprendizado, personalização e performance.

## Métricas

### Justificativas

- Curva de aprendizado
  - D3: a criação dos gráficos é feita de uma maneira mais manual.
  - Chart: possui modelos de gráficos predefinidos e simples de usar.
- Personalização
  - D3: possui grande flexibilidade para criar e personalizar gráficos mais
  complexos.
  - Chart: possui pouca flexibilidade para personalizar os gráficos predefinidos.
- Performance
  - Por ser uma biblioteca mais simples, o Chart é mais leve e rápido em relação ao
  D3.

| Atributos            | D3.js | Chart.js |
| -------------------- | ----- | -------- |
| Curva de aprendizado | Média | Boa      |
| Personalização       | Boa   | Média    |
| Performance          | Média | Boa      |

## Decisão

Chart.js

```

Figura 5: Exemplo de uso do FDRM em linguagem Markdown: código



Contexto

Um projeto frontend utilizando o framework React e que precisa escolher uma biblioteca para mostrar gráficos simples de barra e de área. Esta necessidade surgiu a partir dos requisitos do ticket #1010.

Objetivo

Escolher biblioteca para criação de gráficos.

Opções

D3.js (D3), Chart.js (Chart).

Atributos

Curva de aprendizado, personalização e performance.

Métricas

Justificativas

- Curva de aprendizado
  - D3: a criação dos gráficos é feita de uma maneira mais manual.
  - Chart: possui modelos de gráficos predefinidos e simples de usar.
- Personalização
  - D3: possui grande flexibilidade para criar e personalizar gráficos mais complexos.
  - Chart: possui pouca flexibilidade para personalizar os gráficos predefinidos.
- Performance
  - Por ser uma biblioteca mais simples, o Chart é mais leve e rápido em relação ao D3.

Atributos	D3.js	Chart.js
Curva de aprendizado	Média	Boa
Personalização	Boa	Média
Performance	Média	Boa

Decisão

> Chart.js

Figura 6: Exemplo de uso do FDRM em linguagem Markdown: visualização

3.1.4 Validação

O processo de validação aconteceu em duas etapas. Na primeira etapa foi feita uma apresentação para desenvolvedores de softwares frontend, com o objetivo de mostrar as motivações para criação do modelo FDRM e mostrar os detalhes do mesmo, apresentando seu diagrama de uso (Figura 3) e também um exemplo de uso (Figura 4). Na segunda etapa foi pedido que aos desenvolvedores que respondessem um formulário sobre o modelo apresentado.

3.1.4.1 Formulário

O formulário foi dividido em duas etapas. A primeira etapa objetivou coletar opiniões sobre o modelo sem que os desenvolvedores o tivessem utilizado. Na segunda etapa foi pedido que os desenvolvedores completassem um exemplo pré-definido do modelo, para que tivessem uma experiência prática; e em seguida foram feitas outras perguntadas para saber como foi a experiência de utilizar o modelo. O formulário está no Apêndice A.

3.1.4.2 Perfil dos desenvolvedores que responderam o formulário

O formulário foi respondido por seis desenvolvedores de software, que trabalham em empresas privadas. Seguem abaixo as informações do perfil de cada desenvolvedor (dev):

Nome	Idade	Formação	Anos de experiência	Nível de cargo	Área de atuação
Dev 1	25-30	Sistemas de Informação	7	Pleno	Web: Frontend / Backend
Dev 2	25-30	Análise e Desenvolvimento de Sistemas	6	Pleno	Web: Frontend
Dev 3	40-45	Ciência da Computação	17	Sênior	Web: Frontend / Backend Dados: Eng. de Dados
Dev 4	25-30	Eng. da Computação	3	Júnior	Web: Frontend
Dev 5	30-35	Ciência da Computação	15	Sênior	Web: Frontend / Backend
Dev 6	30-35	Eng. de Software	10	Sênior	Web: Frontend / Backend

Tabela 1: **Perfil dos desenvolvedores que responderam o formulário**

Detalhamento escrito com informações adicionais sobre o tempo aproximado de trabalho em cada área de atuação:

- Dev 1: 25-30 anos, formado em Sistemas de Informação, possui 7 anos de experiência com desenvolvimento de software web, atuando por 4 anos no backend e 3 anos no frontend; atualmente trabalha no cargo de Eng. de Software Pleno;
- Dev 2: 25-30 anos, formado em Análise e Desenvolvimento de Sistemas, possui 6 anos de experiência com desenvolvimento de software web, atuando sempre no frontend; atualmente trabalha no cargo Eng. de Software Pleno;
- Dev 3: 40-45 anos, formado em Ciência da Computação, possui 17 anos de experiência com desenvolvimento de software, atuando por 10 anos no backend de softwares web, 5 anos no frontend de softwares web e 2 anos como Eng. de Dados; já foi líder técnico e projetista; atualmente trabalha no cargo de Eng. de Software Sênior;
- Dev 4: 25-30 anos, formado em Eng. da Computação, possui 3 anos de experiência com desenvolvimento de software web, atuando sempre no frontend; atualmente trabalha no cargo Eng. de Software Júnior;

- Dev 5: 30-35 anos, formado em Ciência da Computação, possui 15 anos de experiência com desenvolvimento de software web, atuando por 7 anos no backend e 8 anos no frontend; atualmente trabalha no cargo de Eng. de Software Sênior;
- Dev 6: 30-35 anos, formado em Eng. de Software, possui 10 anos de experiência com desenvolvimento de software web, atuando por 6 anos no backend e 4 anos no frontend; já foi líder técnico; atualmente trabalha no cargo de Eng. de Software Sênior.

3.2 SUMARIZAÇÃO DO CAPÍTULO

Este capítulo discorreu sobre todo o processo de pesquisa deste trabalho. Inicialmente foi falado sobre as características da pesquisa, informando a natureza, o público, o propósito e a abordagem. Em seguida, detalhou-se sobre o modelo de documentação proposto, o FDRM, explicando sua estrutura e aplicação. Depois, foram dados exemplos sobre o modelo proposto e falou-se sobre o processo de validação do mesmo. E por fim, foram apresentados os perfis dos desenvolvedores que responderam o formulário.

Concluindo, agora que o modelo de documentação proposto foi explicado, o próximo capítulo será focado nos resultados e discussões a respeito das opiniões coletadas sobre o mesmo.

4

RESULTADOS E DISCUSSÕES

Este capítulo disserta sobre os resultados obtidos através do formulário utilizado para coletar opiniões sobre o modelo de documentação proposto neste trabalho.

Como o formulário foi dividido em duas partes, que coletaram respectivamente as opiniões dos desenvolvedores antes e depois da utilização do modelo proposto, este capítulo é estruturado em duas seções, que correspondem a cada parte do formulário.

4.1 ANTES DA UTILIZAÇÃO DO MODELO

4.1.1 Opiniões

Seguem abaixo as opiniões mencionadas pelos desenvolvedores, antes de utilizarem o modelo proposto. Tais opiniões são as percepções iniciais de cada desenvolvedor sobre o que foi apresentado a respeito do modelo.

- Opinião 1: O Dev 1 informou que considerou o modelo bastante interessante, pois "na sua experiência geralmente esse tipo de decisão é tomada de forma arbitrária por alguém, baseada apenas na experiência dessa pessoa (que as vezes é insuficiente para uma tomada de decisão), e então o projeto avança e os problemas começam a aparecer. Às vezes também a decisão é tomada sem considerar, por exemplo, curva de aprendizagem e se ela realmente é a mais adequada para resolução do problema".

- Opinião 2: Dev 2 disse que "a ideia é boa, um ótimo fundamento e ajudaria bastante nas tomadas de decisão relacionadas a qual tecnologia ou biblioteca utilizar no projeto, pois apontando os pontos fortes e fracos na utilização de cada opção, dá-se uma ideia de qual será a melhor decisão baseando-se em atributos chaves para cada projeto em específico".

- Opinião 3: O Dev 3 falou que "achou muito interessante a apresentação desse modelo, principalmente por instituir um padrão na análise de um problema (contexto)". Ele também disse que "achou muito válido e extremamente importante a análise

das variáveis (opções) levando em consideração cada atributo com seu risco ou vantagem".

- Opinião 4: O Dev 4 falou que "gostou do modelo porque ele pode resolver o problema da perda de informações sobre as decisões antigas de um projeto e também pode evitar que alguma tecnologia seja escolhida por motivos passionais".

- Opinião 5: O Dev 5 informou que "achou o modelo interessante para fazer escolhas racionais e que o modelo pode ser utilizado em reuniões de brainstorming sobre as tecnologias que vão ser usadas em um código".

- Opinião 6: O Dev 6 mencionou que "achou o modelo interessante porque vai documentar o contexto, o motivo e o que levou o time a tomar determinada decisão técnica e proporcionar uma base de dados histórica que vai facilitar a tomada de decisões nos projetos".

Todas as percepções convergiram em achar o modelo interessante ou bom, por principalmente proporcionar que uma decisão seja tomada após uma análise do seu contexto e das caracterísicas das opções disponíveis.

Vale destacar o que foi dito pelos devs 4 e 6 no que diz respeito ao modelo proporcionar uma base de dados histórica e poder resolver o problema da perda de informações sobre as decisões antigas de um projeto, pois como o modelo foi pensado para ser utilizado dentro do repositório do software que está sendo desenvolvido, então todos os registros de decisões podem ser facilmente consultados por qualquer desenvolvedor.

Também vale ressaltar o que foi mencionado pelos devs 1 e 4 em relação ao modelo evitar que uma decisão seja tomada de forma arbitrária, apenas com base na experiência de uma pessoa, por motivos até passionais, porque as pessoas naturalmente se acostumam e se apegam às tecnologias que mais utilizaram durante a carreira; e isso pode provocar decisões (não intencionais) menos adequadas as necessidades de um software.

Por fim, pode-se constatar que as percepções iniciais sobre o modelo foram positivas e corroboraram efetivamente com o propósito de registrar de maneira clara as razões que motivam uma determinada tomada de decisão. A seguir serão apresentadas as sugestões de melhoria para o modelo proposto.

4.1.2 Sugestões

Seguem abaixo as sugestões de melhoria propostas pelos desenvolvedores:

- Sugestão 1: Adicionar pesos aos critérios (atributos) para mostrar quais são os pontos mais importantes a serem considerados na avaliação das métricas e portanto, ajudar na tomada de decisão;

- Sugestão 2: Definir sugestões de métricas para quem for usar o modelo e deixar aberto o uso de qualquer outra métrica que for mais adequada ao projeto;
- Sugestão 3: Mudar a forma das métricas para algo quantitativo, como por exemplo a Escala de Likert, porque utilizando algo quantitativo, a medição do que é ruim, médio e bom seria mais assertiva, pois teriam notas. Com isso, a decisão final poderia ser baseada em um valor, o que tornaria o modelo mais fácil, simples e direto, não deixando dúvidas no resultado;
- Sugestão 4: Definir boas práticas para o modelo, a fim de que seus usuários o utilizem da melhor forma possível; e também disponibilizar casos de uso para guiar a aplicação do modelo para quem nunca o tiver utilizado.

As sugestões 1 e 3, em conjunto, podem tornar o modelo mais matemático e programável, visto que adicionar pesos aos atributos e utilizar métricas quantitativas pode possibilitar que uma decisão seja tomada com base na manipulação de tais valores, como por exemplo, fazer uma escolha com base nos números obtidos através da multiplição dos valores dos pesos dos atributos pelos valores das métricas.

As sugestões 2 e 4 são focadas nas pessoas que vão utilizar o modelo e são interessantes porque é importante que os usuários tenham flexibilidade para definir as métricas como quiserem e também é fundamental que existam exemplos para guiar e tornar clara a aplicação do modelo. Além disso, como tais sugestões focam na divulgação e apresentação do modelo para outras pessoas, elas podem ser aplicadas em um repositório do *Github*, por exemplo.

Por fim, todas as sugestões podem ser utilizadas de forma complementar no modelo de documentação proposto, pois é possível (e não existe impeditivo) que o mesmo seja utilizado de forma adaptada.

4.1.3 Interesse em utilizar o modelo

Os 6 desenvolvedores afirmaram ter interesse em utilizar o modelo proposto.

4.2 DEPOIS DA UTILIZAÇÃO DO MODELO

4.2.1 Opiniões

Apenas um desenvolvedor fez uma comentário adicional sobre sua opinião a respeito do modelo, enquanto os outros desenvolvedores mantiveram a mesma opinião. Segue abaixo a opinião mencionada:

- Opinião 1: O Dev 3 mencionou que "ainda acha o modelo interessante pela definição de um padrão de analise e organização das informações levantadas", porém faz uma

ressalva de "que é necessário ter dedicação de tempo para fazer levantamento de informações, estudo e comparação das opções".

A opinião adicional mencionada pelo Dev 3 está relacionada ao tempo que se leva para criar uma documentação, não informando de maneira explícita que custa demasiado tempo para coletar informações e analisá-las, mas sim pontuando que não se pode ignorar o custo de tempo para criar a documentação utilizando o modelo proposto.

É válido mencionar a conexão entre a opinião do Dev 3 e um dos valores do Manifesto Ágil de Software, que diz *"Software em funcionamento mais que documentação abrangente"*. A relação entre ambos é bastante discutida na comunidade de software, no que se refere ao paradoxo do equilíbrio de desenvolver softwares de maneira ágil e documentar softwares de maneira efetiva.

O modelo de documentação proposto não foi criado com o objetivo de registrar as minúcias do desenvolvimento de um software frontend (como as regras de negócio), mas sim registrar as motivações e justificativas das decisões relacionadas ao design do software, que podem impactar todo o ciclo de vida do código. Logo, na opinião do autor que vos fala, o modelo FDRM não se enquadra no propósito de criar documentações abrangentes.

Por fim, vale ressalvar que é preciso ponderar a importância da documentação para a eficiência da própria agilidade no longo prazo, pois assim como é preciso plantar para colher, talvez seja preciso documentar para ser ágil, mas essa não é uma discussão para este trabalho.

A seguir serão mostradas as percepções dos desenvolvedores sobre suas experiências de uso do modelo proposto.

4.2.2 Experiência sobre o uso

Seguem abaixo as percepções dos desenvolvedores sobre suas experiências ao utilizarem na prática o modelo proposto:

- Percepção 1: O Dev 1 informou que "foi uma experiência boa, descomplicada e simples; e que rapidamente chegou a uma decisão".
- Percepção 2: O Dev 3 falou que ao utilizar o modelo "achou mais organizado fazer as comparações, porém é necessário ter uma dedicação de tempo para fazer as leituras das referências e assim ser mais assertivo na opinião e definição da decisão".
- Percepção 3: O Dev 4 mencionou que "gostou de utilizar o modelo, porque fazer a escolha entre as opções ficou mais simples quando as características de cada uma foram comparadas" e também falou que achou "o fluxo do modelo tranquilo de desenvolver".
- Percepção 4: O Dev 6 mencionou que "foi uma boa experiência, pois o forçou a pesquisar mais a fundo o comportamento e as diferenças das opções disponíveis".

- Os devs 2 e 5 não utilizaram o modelo, pois tiveram imprevistos para finalizar o preenchimento do formulário.

De maneiral geral, as percepções foram positivas porque a maioria dos respondentes afirmou ter tido uma boa experiência ao utilizar o modelo.

O Dev 1 informou que achou o modelo descomplicado, simples e rápido para chegar a uma decisão, o que foi ao encontro do que o Dev 4 mencionou sobre a simplicidade de fazer uma escolha após ter comparado as características das opções disponíveis e também ter achado o fluxo do modelo tranquilo de desenvolver.

O Dev 6 informou ter tido uma boa experiência porque o modelo o forçou a investigar em detalhes as diferenças entre as opções de escolha disponíveis. Tal comentário pode ser entendido de forma contraditória, porque as vezes as pessoas tem resistência para fazer algo que lhes coloque numa posição de buscar informações adicionais, visto que demanda esforço e tempo, como mencionou o Dev 3. No entanto, cabe sempre lembrar, por exemplo, que criar softwares para indústria é um trabalho que exige (ou deveria exigir) responsabilidade e compromisso com o que está sendo feito, pois custa muito caro e portanto é fundamental investigar e entender os detalhes das coisas que estão sendo decididas.

A seguir serão mostradas as respostas dos desenvolvedores ao utizarem o modelo proposto, apresentando como cada respondente tomou a decisão que foi solicitada no formulário.

4.2.3 Respostas da decisão que foi solicitada no formulário

Assim como pode ser visto no Apêndice A, a decisão que foi solicitada no formulário pede que os respondentes decidam entre as opções *CSS-in-JS* e *pré-processadores CSS*, em uma decisão cujo objetivo é escolher uma abordagem de estilização dos componentes de um projeto frontend utilizando o framework *React.js*.

Os respondentes chegaram a diferentes decisões por diferentes justificativas. Seguem abaixo as respostas:

Detalhes da decisão que você precisa tomar

Contexto: projeto front-end de uma aplicação bancária utilizando o framework React.js.

Objetivo: escolher abordagem de estilização dos componentes.

Opções: CSS-in-JS , pré-processadores CSS.

Lista de atributos: manutenibilidade, desempenho, compatibilidade com bibliotecas

Métricas: Classificar as opções como *Bom*, *Médio* ou *Ruim*, para cada atributo. Por favor, se possível, adicionar justificativas.

Atributos	CSS-in-JS	pré-processadores CSS
manutenibilidade	Bom	Médio
desempenho	Médio	Bom
compatibilidade com bibliotecas	Ruim	Boa

Decisão: Como meu projeto tende a crescer, precisa escalar e o desempenho é crucial nele, baseado no preenchimento do modelo, utilizar pré processadores CSS é o mais adequado.

Figura 7: Resposta do Dev 1 sobre a decisão que foi solicitada no formulário

Eu usaria o "pré-processadores CSS" por ter mais conhecimento e experiência em relação a outra tecnologia. Mas confesso que não consegui fazer muitas leituras e comparações.

Figura 8: Resposta do Dev 3 sobre a decisão que foi solicitada no formulário

Contexto: projeto frontend utilizando o framework React.js.

Objetivo: escolher abordagem de estilização dos componentes.

Opções: CSS-in-JS , pré-processadores CSS.

Atributos: Curva de aprendizado, Escalabilidade e Performance.

Métricas

Justificativas

- Curva de aprendizado
  - CSS-in-JS: é preciso aprender a sintaxe de estilização utilizando JS, que é diferente da sintaxe do CSS.
  - pré-processadores CSS: como usa uma sintaxe similar ou igual ao CSS, é mais simples de aprender.
- Escalabilidade
  - CSS-in-JS: a estilização baseada em componentes evita que estilos sejam sobrepostos, impedindo problemas com escopo global de classes.
  - pré-processadores CSS: necessita de maior disciplina, padronização e/ou adoção de metodologias e ferramentas para ser escalável e evitar problemas com escopo global de classes.
- Performance
  - CSS-in-JS: estilos são renderizados apenas se os componentes estiverem em tela; permite otimizações dos bundlers javascript; pode apresentar problemas de delay e cache no navegador.
  - pré-processadores CSS: são baixados arquivos de componentes não renderizados em tela.

Atributos	CSS-in-JS	pré-processadores CSS
Curva de aprendizado	Média	Boa
Escalabilidade	Boa	Média
Performance	Média	Média

Decisão: **CSS-in-JS**

Figura 9: Resposta do Dev 4 sobre a decisão que foi solicitada no formulário

Contexto: projeto frontend utilizando o framework React.js.

Objetivo: escolher abordagem de estilização dos componentes.

Opções: CSS-in-JS , pré-processadores CSS.

Atributos: Reuso de estilos, Compatibilidade com navegadores.

Métricas

Justificativas

- Reuso de estilos
  - CSS-in-JS: Se você usa alguma biblioteca, tipo a styled-components, é fácil estender um componente criado e customizar sua estilização em pontos específicos, além disso ele gera o css na renderização do componente, baixando pro usuário só que é necessário pros componentes que estão sendo renderizados.
  - pré-processadores Para customizar um estilo para um componente específico é mais difícil, deixa a leitura mais difícil do código e é necessário associar os estilos a IDs e classes.
- Compatibilidade com navegadores
  - CSS-in-JS: Por gerar o css em tempo de execução, a biblioteca gera o css que seja compatível com o navegador que está sendo utilizado, o programador não precisa ter essa preocupação.
  - pré-processadores CSS: Manter a compatibilidade com diversos navegadores depende de implementação do desenvolvedor.

Atributos	CSS-in-JS	pré-processadores CSS
Reuso de estilos	Bom	Média
Compatibilidade com navegadores	Bom	Média

Decisão: **CSS-in-JS**

Figura 10: Resposta do Dev 6 sobre a decisão que foi solicitada no formulário

Pode-se notar que as respostas acima chegaram a diferentes decisões e foram baseadas em diferentes atributos. O que demonstra que mesmo que o contexto da decisão seja igual, desenvolvedores diferentes podem fazer escolhas diferentes.

Sendo assim, as diferentes respostas acima reforçam a necessidade de criar documentações que registrem tomadas de decisões, porque os futuros mantenedores de um software podem não compreender alguns detalhes do mesmo, por simplesmente terem uma perspectiva diferente daqueles que estavam desenvolvendo o software antes.

Sobre o formato das respostas, as Figuras 9 e 10 foram formatadas pelo autor que vos fala, porque as respostas foram enviadas em formato de texto.

É importante pontuar que a resposta da Figura 8 vai totalmente de encontro com o propósito do modelo de documentação proposto neste trabalho, porque a decisão está sendo baseada apenas na experiência do desenvolvedor, sem nenhuma justificativa ou argumento que

explique porque uma opção é mais adequada que a outra. Vale lembrar que este problema foi mencionado que pelos devs 1 e 4, na seção 4.1.1.

Por fim, é de extrema importância pontuar que não cabe ao modelo definir se as respostas acima estão certas ou erradas, mas sim fornecer uma estrutura para que todas as motivações e justificativas das decisões sejam devidemente detalhadas, para que dessa forma fiquem claros os fatores que levaram à determinada decisão.

4.3 SUMARIZAÇÃO DO CAPÍTULO

Este capítulo apresentou e discorreu sobre os resultados obtidos através do formulário utilizado para coletar opiniões de engenheiros de software sobre o modelo FDRM. Os resultados foram separados em duas seções, respectivas aos momentos de antes e depois dos respondentes terem utilizado o modelo; e foram estruturados nas seções de opiniões, sugestões, interesse de utilização e percepções das experiências de uso do modelo.

Por fim, como sequência aos resultados e discussões apresentados, o próximo capítulo apresentará as considerações finais sobre este trabalho.

5

CONSIDERAÇÕES FINAIS

Este capítulo apresenta as conclusões obtidas através dos resultados deste trabalho e descreve limitações do trabalho.

5.1 CONCLUSÃO

Os resultados demonstraram um bom indício de efetividade de que o modelo de documentação proposto pode registrar decisões de forma estruturada, coesa e simples de criar e compreender.

As experiências de uso mostraram que é pré-requisito implícito o estudo das opções e atributos presentes no modelo, para que as justificativas sejam adequadas e coerentes com o contexto da decisão.

Vale ressaltar que por utilizar as métricas *Bom, Médio e Ruim*, o modelo apresenta um viés de comparação qualitativo e não quantitativo, o que facilitaria sua implementação como um algoritmo, pois as escolhas poderiam ser feitas com base em números.

Algumas sugestões de melhoria foram propostas para tornar o modelo mais matemático, programável e também ajudar a divulgar o modelo para outras pessoas.

Todos os respondentes disseram ter interesse em utilizar o modelo proposto, visto que sua utilização possibilita gerar documentos que ajudam os desenvolvedores a terem um entendimento mais completo das ferramentas utilizadas em um software.

As diferentes respostas na decisão que foi solicitada no formulário indicaram que existem divergências em decisões tomadas por diferentes desenvolvedores, mesmo que o contexto das decisões seja o mesmo. Portanto, se tais decisões não forem registradas, é possível que os futuros mantenedores de um projeto tenham dificuldade para entendê-lo. Por esses motivos, o modelo FDRM pode ser ferramenta útil para os processos de documentar e tomar decisões.

Por fim, este trabalho conseguiu ser efetivo em apresentar uma estratégia para tomadas de decisão e iniciar as investigações e avaliações sobre a mesma.

5.2 LIMITAÇÕES DO TRABALHO

Este trabalho se limitou a documentar softwares frontend, e portanto, foi validado com desenvolvedores de software frontend. No entanto, é importante ressaltar a forte relação entre softwares frontend e backend, que atualmente são desenvolvidos em forte sincronia. Como exemplo disso, existe a estratégia *Backend For Frontend* (BFF), que objetiva, entre outras coisas, a criação de um backend focado nas experiências do usuário (no frontend). Portanto, este trabalho poderia se estender a softwares backend também.

Vale ressaltar que o tempo foi um fator limitante para aprimoramentos do modelo e experimentações das sugestões propostas pelos desenvolvedores que responderam o formulário, pois é uma possibilidade aplicar o que foi dito em algumas sugestões e tornar o modelo proposto mais robusto. Além disso, caso houvesse mais tempo para desenvolver o trabalho, teria sido possível validar o modelo proposto com mais desenvolvedores, e portanto, garantir maior credibilidade e robustez sobre os resultados obtidos.

Por fim, é importante mencionar que poderia ser feita uma comparação entre o modelo proposto neste trabalho e alguns modelos que aparecem na Figura 1. Esta é sem dúvidas uma falta neste trabalho, que ficaria mais completo caso não a tivesse. No entanto, tal comparação também é uma possibilidade para trabalhos futuros.

5.3 TRABALHOS FUTUROS

Seguem abaixo algumas possibilidades possíveis para trabalhos futuros:

- Investigar se o estilo de documentação proposto neste trabalho também auxilia outros tipos de software além do web frontend;
- Criar um software para implementar o modelo proposto;
- Aprimorar o modelo proposto utilizando as sugestões obtidas no formulário ou qualquer outra proposta de melhoria;
- Comparar o modelo FDRM com outros modelos de ADR.

REFERÊNCIAS

[1] Aquino, C. & Gandee, T. (2016). *Front-End Web Development: The Big Nerd Ranch Guide*. Big Nerd Ranch Guides; 1ł edição (29 julho 2016), 1. ed edition.

[2] Baskerville, R., Ramesh, B., Levine, L., Pries-Heje, J., & Slaughter, S. (2003). Is "internet-speed"software development different? *IEEE Software*, 20(6):70–77.

[3] Belady, L. A. & Lehman, M. M. (1976). A model of large program development. *IBM Systems Journal*, 15(3):225–252.

[4] Berander, P. & Jönsson, P. (2006). A goal question metric based approach for efficient measurement framework definition. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, 316325.

[5] (by webfoundation.org), W. W. W. F. (2019). 30 years on, whats next fortheweb? https://webfoundation.org/2019/03/web-birthday-30/. [Online; accessed 29-April-2023].

[6] Chart.js (2023). Home page. https://www.chartjs.org/. [Online; accessed 28-February-2023].

[7] D3.js (2023). Home page. https://d3js.org/. [Online; accessed 28-February-2023].

[8] da Silva, C. V. P., de Castro Campos, D., de Moura, D. C., & Nery, P. (2009). Gqm. https://www.cin.ufpe.br/~scbs/metricas/seminarios/GQM_texto.pdf. [Online; accessed 07-March-2023].

[9] Eis, D. (2015). *Guia Front-End. O Caminho das Pedras Para Ser Um Dev Front-End*. Casa do Código (1 janeiro 2015), 1. ed edition.

[10] Henderson, J. P. & Contributors (2023). Architecture decision record (adr). https://github.com/joelparkerhenderson/architecture-decision-record. [Online; accessed 05-March-2023].

[11] (https://github.com/adr), A. D. R. G. R. (2023). Markdown any decision records. https://adr.github.io/madr/. [Online; accessed 15-April-2023].

[12] Kopp, O., Armbruster, A., & Zimmermann, O. (2018). Markdown architectural decision records: Format and tool support. In *Central-European Workshop on Services and their Composition*.

[13] Len Bass, P. C. & Zazman, R. (2021). *Software Architecture in Practice*. Addison-Wesley Professional; 4th edition (August 3, 2021), 4. ed edition.

[14] Lethbridge, T., Singer, J., & Forward, A. (2003). How software engineers use documentation: the state of the practice. *IEEE Software*, 20(6):35–39.

[15] (mozilla.org), M. F. (2023). Javascript. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. [Online; accessed 28-February-2023].

[16] Nygard, M. (2011). Documenting architecture decisions. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. [Online; accessed 29-April-2023].

[17] Olaf Zimmermann, Doc SoC, Z. B. (2021a). Adr = any decision record? https://medium.com/olzzio/adr-any-decision-record-916d1b64b28d. [Online; accessed 23-March-2023].

[18] Olaf Zimmermann, Doc SoC, Z. B. (2021b). From architectural decisions to design decisions. https://medium.com/olzzio/from-architectural-decisions-to-design-decisions-f05f6d57032b. [Online; accessed 23-March-2023].

[19] Overflow, S. (2021). 2021 developer survey. https://insights.stackoverflow.com/survey/2021/. [Online; accessed 28-February-2023].

[20] Overflow, S. (2022). 2022 developer survey. https://survey.stackoverflow.co/2022/. [Online; accessed 28-February-2023].

[21] Parnas, D. L. (2011). *Precise Documentation: The Key to Better Software*, 125–148. Springer Berlin Heidelberg, Berlin, Heidelberg.

[22] React (2023). Introdução. https://pt-br.reactjs.org/docs/getting-started.html. [Online; accessed 28-February-2023].

[23] SDK, C. (2023). Architecture decision records (adr). https://docs.cosmos.network/main/architecture. [Online; accessed 03-March-2023].

[24] Stark, G., Durst, R., & Vowell, C. (1994). Using metrics in management decision making. *Computer*, 27(9):42–48.

[25] turing.com (2023). What are top web development challenges and solutions to tackle them? https://www.turing.com/resources/top-web-development-challenges#top-8-challenges-in-web-development-and-their-solutions. [Online; accessed 29-April-2023].

[26] Valente, M. T. (2022). *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Independente; Primeira Edição (Janeiro 1, 2022), 1. ed edition.

[27] van Solingen (Revision), R., Basili (Original article, 1994 ed.), V., Caldiera (Original article, 1994 ed.), G., & Rombach (Original article, 1994 ed.), H. D. (2002). *Goal Question Metric (GQM) Approach*. John Wiley Sons, Ltd.

[28] Weare, C. & Lin, W.-Y. (2000). Content analysis of the world wide web: Opportunities and challenges. *Social Science Computer Review*, 18(3):272–292.

[29] Zimmermann, O., Wegmann, L., Koziolek, H., & Goldschmidt, T. (2015). Architectural decision guidance across projects - problem space modeling, decision backlog management and cloud computing knowledge. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*, 85–94.

A

FORMULÁRIO SOBRE O MODELO PROPOSTO DE DOCUMENTAÇÃO DE SOFTWARE FRONTEND

Formulário sobre o modelo de documentação de software frontend apresentado.

Oi, tudo bem?

Sou Rosinaldo Guedes, criador deste formulário.

Este formulário faz parte do meu Trabalho de Conclusão de Curso (TCC) do CIn - UFPE e tem o objetivo de obter sua opinião sobre o modelo de documentação que apresentei para você.

Não é objetivo deste formulário medir conhecimento nem fazer qualquer coisa relacionada à isto.

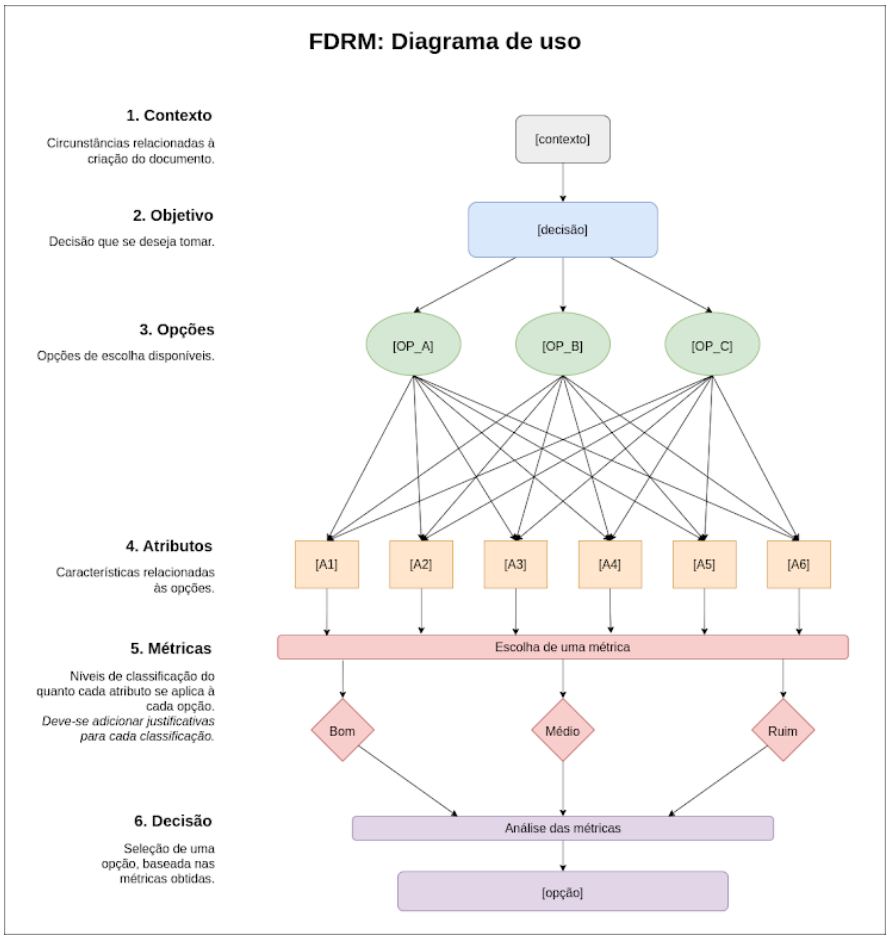
Este formulário tem intenção de apenas coletar a opinião de desenvolvedores de software frontend reais sobre o modelo de documentação apresentado.

Nenhum nome será citado no TCC, apenas as respostas.

Muito obrigado pela ajuda.

*Obrigatório

As pergunta são referentes ao modelo de documentação da imagem abaixo, o mesmo que foi apresentado.



1. Qual a sua opinião sobre o modelo apresentado? Por favor, se houverem, cite pontos positivos e/ou negativos. (Toda opinião, em relação qualquer perspectiva, é relevante.) *

2. Você tem sugestões para o modelo? *

3. Você tem interesse em utilizar o modelo no futuro? *

Marque todas que se aplicam.

- [ ] Sim
- [ ] Não

Utilização do modelo

Na apresentação, mostrei um exemplo preenchido do modelo, para servir de exemplo.
A intenção agora é você mesmo tentar utilizar o modelo, para ter uma noção de como funciona na prática.

Para relembrar, segue um exemplo preenchido:

Contexto: Um projeto frontend utilizando o framework React e que precisa escolher uma biblioteca para mostrar gráficos simples de barra e de área. Esta necessidade surgiu a partir dos requisitos do ticket #1010.

Objetivo: Escolher biblioteca para criação de gráficos.

Opções: D3.js (D3), Chart.js (Chart).

Atributos: Curva de aprendizado, personalização e performance.

Métricas

Justificativas

- Curva de aprendizado
  - D3: a criação dos gráficos é feita de uma maneira mais manual.
  - Chart: possui modelos de gráficos predefinidos e simples de usar.
- Personalização
  - D3: possui grande flexibilidade para criar e personalizar gráficos mais complexos.
  - Chart: possui pouca flexibilidade para personalizar os gráficos predefinidos.
- Performance
  - Por ser uma biblioteca mais simples, o Chart é mais leve e rápido em relação ao D3.

Atributos	D3.js	Chart.js
Curva de aprendizado	Média	Boa
Personalização	Boa	Média
Performance	Média	Boa

Decisão: **Chart.js**

Referências

- https://marquesfernandes.com/desenvolvimento/6-bibliotecas-javascript-para-criacao-de-graficos/
- https://medium.com/@mintholic1/chart-js-vs-d3-js-45e9abe1f08c
- https://www.createwithdata.com/d3js-or-chartjs/

Detalhes da decisão que você precisa tomar

Contexto: projeto frontend utilizando o framework React.js.

Objetivo: escolher abordagem de estilização dos componentes.

Opções: CSS-in-JS , pré-processadores CSS.

Lista de atributos: [colocar lista de atributos aqui].

Não tem um número mínimo nem máximo.

Métricas: Classificar as opções como *Bom*, *Médio* ou *Ruim*, para cada atributo. Por favor, se possível, adicionar justificativas.

Para ser mais rápido, não precisa criar uma tabela, ela serve apenas para facilitar a visualização. Você pode indicar as classificações de outra forma.

Decisão: [colocar decisão aqui].

Abaixo, existem duas caixas de resposta, uma para responder em formato de **texto** e outra para importar um **arquivo**, caso prefira fazer em outro lugar. Escolha a que for melhor para você.

4. Resposta em texto

5. Resposta em arquivo

   Arquivos enviados:

6. Como foi a sua experiência ao utilizar o modelo? *

7. Sua opinião sobre o modelo continua a mesma? Caso algo tenha mudado, por favor, explicar. *

Muito obrigado pela sua ajuda!