FUiDD: Desenvolvimento Dirigido à Funcionalidade de Interface de Usuário

Raíssa Versolatto Faccioli, André Constantino da Silva

[1]Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação
Instituto Federal de São Paulo (IFSP) - câmpus Hortolândia
Avenida Thereza Ana Cecon Breda, N.º 1896, Vila São Pedro
Hortolândia – SP – Brasil

raissafaccioli@outlook.com, andre.constantino@ifsp.edu.br

Abstract. *This paper presents Feature User Interface Driven Development, which is a sequence of activities for the design, development, testing and deployment of user interface components. This research started from the lack of agile methodologies applied to the user interface, resulting in the lack of standardization of commonly used processes. Thus, the five FUiDD processes were built from Feature Driven Development for the development of user interfaces using component-based frameworks. Finally, to validate FUiDD activities, a case study was applied developing a prototype e-Commerce application.*

Resumo. *Este trabalho apresenta o Feature User Interface Driven Development (Desenvolvimento Dirigido à Funcionalidade de Interface de Usuário, traduzido para o Português), uma sequência de atividades para a concepção, desenvolvimento, testes e implantação de componentes de interface de usuário. Esta pesquisa teve início a partir da ausência de metodologias ágeis aplicadas à interface de usuário, resultando na carência de padronização dos processos comumente utilizados. Sendo assim, os cinco processos do FUiDD foram baseados no Feature Driven Development para o desenvolvimento de interfaces de usuário utilizando frameworks baseados em componentes. Por fim, para validar as atividades do FUiDD, foi realizado um estudo de caso desenvolvendo um protótipo de aplicação e-Commerce.*

1. Introdução e Motivação

As metodologias ágeis tornaram-se populares nos processos de concepção e desenvolvimento de softwares nos últimos anos, facilitando, entre outros fatores, a colaboração com os clientes, tempo de resposta para mudanças de requisitos e a entrega contínua de incrementos operacionais valiosos de software [Beck et al. 2001]. Consequentemente, a arquitetura de software dos projetos realizados na área de desenvolvimento web adequaram-se a fim de atenderem com mais eficiência as exigências propostas pelas metodologias ágeis. Grandes empresas como Netflix e Amazon adotaram os microsserviços, por exemplo, como arquiteturas para suas aplicações.

Os microsserviços tratam-se de uma abordagem de arquitetura de sistemas que se baseia no conceito de modularização, no qual cada microsserviço é implementado e operado como um pequeno, mas independente sistema, oferecendo acesso à sua lógica

e dados internos por meio de uma interface de rede [Jamshidi et al. 2018]. A utilização de arquiteturas modulares provém às aplicações, entre outras vantagens, maior escalabilidade, resiliência e heterogeneidade tecnológica [Thönes 2015].

As mesmas mudanças ocorreram no âmbito do desenvolvimento de interfaces de usuário, nos quais os *frameworks* de desenvolvimento *front-end* baseados em componentes ganharam notoriedade nas aplicações *web*. *Frameworks* como o *Angular*, *Vue.js* e *Reactive* possuem respectivamente 419.192, 1.049.397 e 5.340.239 downloads no NPM [npm 2019] [1] no mês de setembro do ano de 2019, como ilustra a Figura 1:

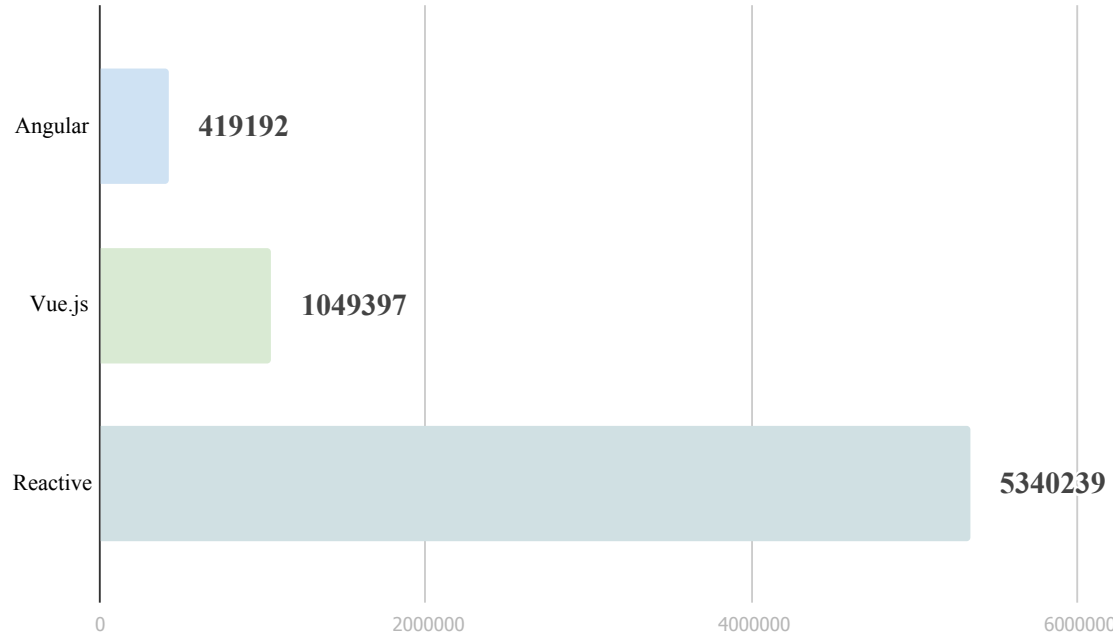

Figura 1. *Downloads* dos pacotes *Angular*, *Vue.js* e *Reactive* via NPM em Setembro de 2019

Apesar das tendências de desenvolvimento *back-end* e *front-end* estarem alinhadas em relação à modularização e desacoplamento de arquiteturas através da utilização de componentes [Oh et al. 2017], há falta de divulgação de metodologias ágeis cujo foco seja o processo de desenvolvimento de interfaces de usuário (UIs), de acordo com as pesquisas realizadas. Sendo assim, o objetivo dessa pesquisa é propor um modelo de processos baseados no *Feature Driven Development*, visando o *design*, desenvolvimento, testes e implantação de componentes de interface de usuário, chamado de *Feature User Interface Driven Development* (FUiDD).

2. Referencial teórico

Para complementar o processo FUiDD criado nessa pesquisa, é preciso compreender alguns conceitos e metodologias utilizados. Sendo assim, o referencial necessário consiste

[1] Traduzido do inglês, o Gerenciador de Pacotes Node é o maior registrador de software atualmente, contendo mais de 800.00 pacotes de códigos [npm 2019]

na compreensão do FDD e do conceito de componentes de interface de usuário, explorados nas seções a seguir.

2.1. *Feature Driven Development* (FDD)

Criado por Peter Coad, Jeff De Luca e Stephen Palmer, o Desenvolvimento Orientado à Funcionalidades (*Feature Driven Development*, traduzido do Inglês, ou ainda FDD) é um modelo de processos de projeto ágil cujo software é desenvolvido de acordo com funcionalidades de valor para o cliente [Shabib Aftab et al. 2018]. O FDD possui cinco processos, como mostrado na Figura 2, e que são explicados a seguir.

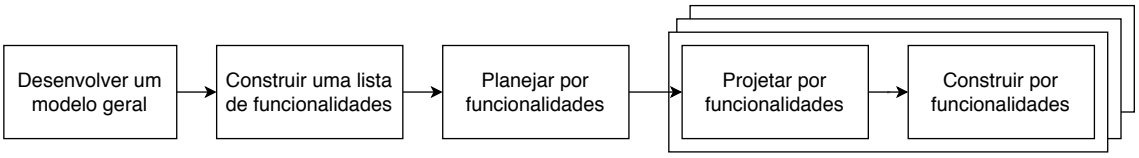

Figura 2. Processos do FDD. Traduzido de Jeff De Luca (1999)

O primeiro processo, "Desenvolver um modelo geral", refere-se a criação de uma representação de alto nível do software a ser implementado. Para tanto, os membros da equipe de desenvolvimento recebem uma explicação do escopo e do contexto do sistema advindas do líder do projeto. Posteriormente, o grupo de desenvolvimento particiona o software em diferentes domínios de negócio, sendo estes áreas comumente utilizadas no mercado em geral e que geram funcionalidades dentro do software, como por exemplo Compras e Vendas [Palmer and Felsing 2001]. Os domínios são especificamente criados para cada projeto, adequando nomenclaturas e processos relacionados à lógica de negócio a ser implementada. Por fim, os desenvolvedores produzem os modelos gerais de acordo com os domínios de negócio criados. Geralmente são criados diagramas de caso de uso, de sequência e de atividade.

No segundo processo, "Construir uma lista de funcionalidades", cada modelo geral criado nos domínios é refinado com mais detalhes e, a partir do mesmo, é gerada uma lista de funcionalidades a serem implementadas. Ao final desse processo, são obtidos todos os requisitos funcionais do software agrupados por seus domínios de negócio [Firdaus et al. 2014].

O terceiro processo, "Planejar por funcionalidades", refere-se à ordem em que as funcionalidades devem ser implementadas. Dessa forma, o líder do projeto, juntamente com o dono do software (o cliente), as organiza e prioriza. Posteriormente, o líder de projeto faz a designação de cada funcionalidade para o respectivo desenvolvedor, combinando a capacidade da equipe de projeto com a quantidade de requisitos a serem implementados.

Por fim, no processo "Construir por funcionalidades", o desenvolvedor testa o código criado no processo anterior, utilizando testes unitários e de integração. Posteriormente, a funcionalidade é validada com o cliente e, se atendidos os requisitos de qualidade e de definição de pronto, a mesma é liberada para implantação [Goyal 2008].

2.2. Componentes de Interface de Usuário

[Harms et al. 2017] analisaram possíveis arquiteturas de interfaces de usuário para a integração com microsserviços. Dentre as opções estudadas, há a interface de usuário mo-

nolítica composta pelos chamados fragmentos ou componentes UI, ressaltados pelos autores como unidades de design gráfico que podem ser reutilizadas em diversos projetos e, quando unidas, constituem a interface visualizada e navegada pelo usuário final. Esses componentes podem, se necessário, estabelecer conexões com os respectivos microsserviços, totalizando a lógica de navegação e a interação com os sistemas de *back-end* provedores de dados. O conceito de componente definido pelos pesquisadores será utilizado nesta pesquisa, sendo apresentado nas sessões seguintes. A Figura 3 ilustra a arquitetura descrita:

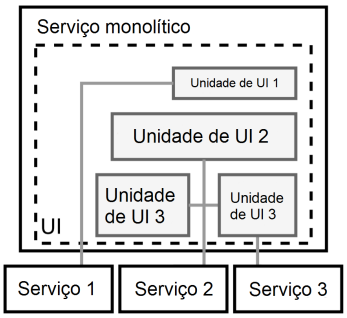

Figura 3. Esquema de composição de interface de usuário baseada em componentes. Traduzido de Harms (2017)

3. Trabalhos correlatos

[Hasselbring and Steinacker 2017] apresentam o conceito de verticais de sistemas, definidas como decomposições auto contidas de software geradas a partir de um conjunto de funcionalidades, cuja origem é a análise e a especificação dos requisitos. Essas funcionalidades são, primeiramente, agrupadas por afinidade e proximidade dado a lógica de negócio do projeto. A partir das mesmas, são geradas as especificações das atividades a serem feitas, desde o desenvolvimento até a implantação. Nesta pesquisa não é definida a diferenciação entre *back-end* ou *front-end* durante a definição das atividades, entretanto as verticais inspiraram alguns processos utilizados no FUiDD. Segundo os autores, os times de desenvolvedores do projeto são responsáveis pelo design, desenvolvimento, implantação e manutenção de cada vertical, sendo estas designadas aos membros da equipe pelo líder do projeto. Ao final do processo, cada vertical do sistema transforma-se em um componente auto sustentável.

[Shabib Aftab et al. 2018] criaram um processo análogo ao *Feature Driven Development* chamado de *Simplified Feature Driven Development* (traduzido do Inglês como "Desenvolvimento Simplificado Orientado a Funcionalidade", ou, ainda, SFDD), cuja metodologia e objetivos assemelham-se com esta pesquisa. A diferença consiste no objetivo de ambos os processos. Enquanto o SFDD foi criado com foco nas mudanças de requisitos em um projeto, o FUiDD foca no desenvolvimento de componentes de interface de usuário. Segundo os pesquisadores, o SFDD foi desenvolvido para suprir a necessidade das mudanças de requisitos no escopo de projetos. A Figura 4 detalha quais são os processos criados no SFDD e quais são as principais tarefas definidas para cada um deles. Além disso, a pesquisa evidencia que os modelos de processos do *Feature Driven Development* são mais proveitosos em projetos de grande porte, cujas equipes sejam numerosas e o

escopo abrangente. Segundo os autores, isso se deve pelo número relativamente grande de papéis a serem exercidos nos processos do FDD e a dependência direta com desenvolvedores que já tenham experiência prévia com metodologias de desenvolvimento ágeis, fatores que serão utilizados como premissas para esta pesquisa.

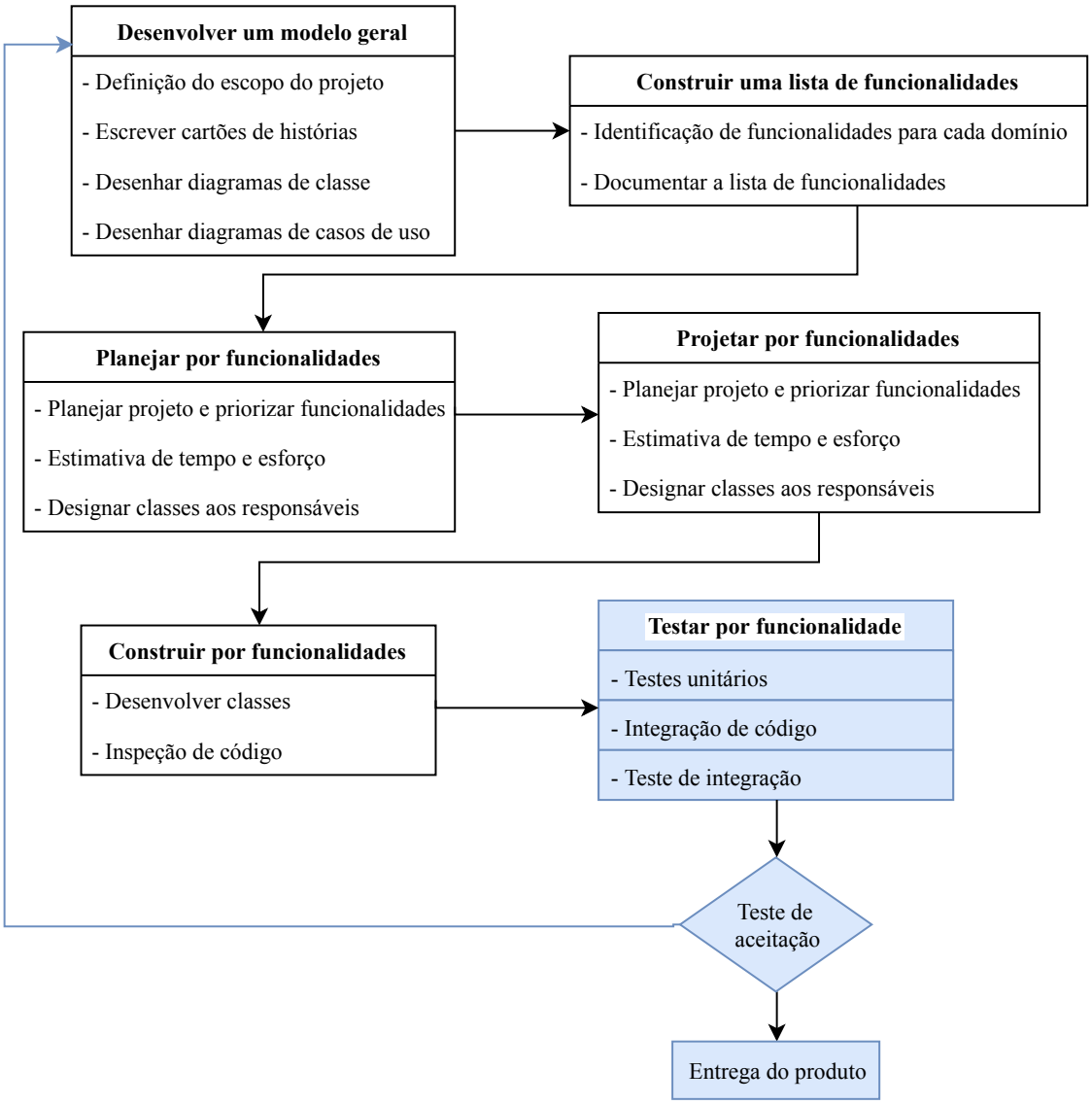

Figura 4. Processo do *Simplified Feature Driven Development*. Traduzido de Shabib Aftab (2018)

Analogamente, [Thakur and Singh 2014] utilizaram o *Feature Driven Development* como fundamento para criar um processo análogo, chamado *Feature Driven Reuse Development* (FDRD), visando adicionar ao FDD a funcionalidade de reuso de software, mostrando, ao ser implementado, melhoras na produtividade da organização e na qualidade dos produtos desenvolvidos. Segundo os autores, o conceito de reuso de software consiste em um processo de criação de um novo software a partir de artefatos já existentes, que podem ser padrões de design, código, casos de teste, documentações, conhecimentos e *frameworks*. A Figura 5 mostra a sequência de processos do FDRD, em que foram mantidos os cinco processos originais do FDD e adicionados dois subprocessos em "Desen-

volver um modelo geral". Tais subprocessos, "Novos requisitos do projeto"e "Conjunto de funcionalidades reutilizáveis", referem-se na escolha e classificação de quais componentes de software (incluindo a interface de usuário e o *back-end*) podem ser reutilizáveis no projeto, utilizando para tanto métricas de classificação criada pelos autores.

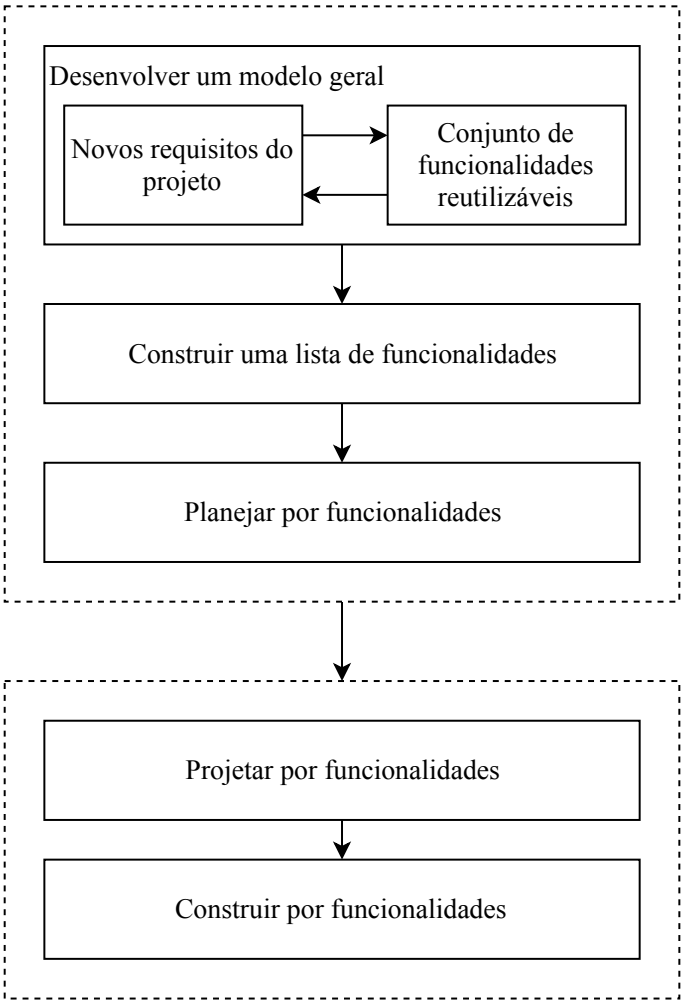

Figura 5. Processos do *Feature Driven Reuse Development*. Traduzido de Thankur e Singh (2014)

4. Materiais e métodos

O estudo iniciou-se com a pesquisa bibliográfica realizada nos banco de dados acadêmicos ACM e IEEE. Para tanto, foram buscadas as palavras chaves "*feature driven development*", "*feature driven development ui*", "*ui feature based*", "*user interface components*"e "*fdd related process*"para pesquisas a partir do ano de 2010, cuja pesquisa resultou em alguns artigos em que pesquisadores criaram processos análogos ou baseados no FDD, adicionando ao mesmo novas atividades a fim de suprir alguma necessidade característica de cada pesquisa. Tais artigos foram apresentados na Seção 3. Após encontrados os materiais referentes ao FDD, foram estudados os seus cinco processos e suas principais atividades, como mostrados na Seção 3.

Para a criação dos processos do FDD aplicado ao desenvolvimento de interfaces de usuário, foram levantados e organizados hierarquicamente os principais processos

comumente realizados durante a concepção de uma interface de usuário. Por fim, tais processos foram modelados de acordo com os processos do FDD. Para testar os processos criados, foi aplicado um estudo de caso de uma página de *e-Commerce* criada a partir de um conjunto de requisitos.

5. *Feature User Interface Driven Development*

A adaptação dos processos do FDD foi moldada a fim de aplicar seus processos ao desenvolvimento de interfaces de usuário (do inglês "*User Interface*", cuja sigla é UI). Para tanto, cada processo original do FDD foi a base para a adaptação de processos comumente utilizados durante a projeção e desenvolvimento de uma UI [Harms et al. 2017]. A Tabela 1 apresenta a comparação entre os cinco processos do FDD e do FUiDD:

Feature Driven Development	***Feature User Interface Driven Development***
Desenvolver um modelo geral	Prototipar UI
Construir uma lista de funcionalidades	Especificar componentes na UI
Planejar por funcionalidades	Planejar por componentes de UI
Projetar por funcionalidades	Projetar por grupos de componentes de UI
Construir por funcionalidades	Construir por grupos de componentes de UI

Tabela 1. Comparação entre os processos *Feature Driven Development* e *Feature User Interface Driven Development*

A Figura 6 apresenta a sequência dos cinco processos do *Feature User Interface Driven Development*, sendo os dois últimos processos iterativos. Cada um deles será descrito a seguir, e, para demonstrá-los, será utilizado um estudo de caso de uma página *e-Commerce* descrito na Seção 6.

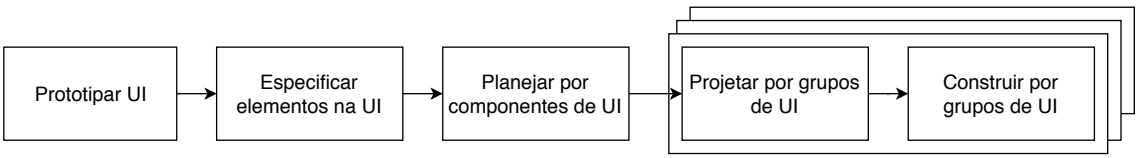

Figura 6. Processos do *Feature User Interface Driven Development*

6. Estudo de caso

Para testar os processos do *Feature User Interface Driven Development* e demonstrar seu funcionamento, foi aplicado como estudo de caso uma página de visualização de um produto de um *e-Commerce*. O estudo de caso tem o intuito de simular parte um projeto de grande porte, contemplando o design da aplicação, cujas atividades foram realizadas até o momento da entrega do documento de especificação de design da interface do usuário e das *APIs* de integração. Sendo assim, serão apresentados os requisitos funcionais e em seguida aplicados os processos do *FUiDD*.

6.1. Requisitos funcionais

1. A página de especificação do produto deverá possuir uma foto grande do produto, seu nome, sua descrição, outras fotos e seu preço;

2. É necessário mostrar opções de pagamento e uma indicação para comprar o produto de acordo com a opção selecionada;
3. Essa página deverá disponibilizar uma área para cálculo de frete a partir do CEP que o usuário fornecer.

6.2. Prototipar UI

A prototipação trata-se da representação dos aspectos do software que serão visíveis ao usuário final, ou seja, o layout da interface de usuário [Pressman 2005]. Sendo uma ferramenta bastante conhecida na Engenharia de Software e nas metodologias ágeis devido à sua facilidade de implementação e baixo custo, a prototipação é definida como o primeiro processo do FUiDD, sendo um objeto geral do software na perspectiva do desenvolvimento *front-end*.

Sendo assim, dada a descrição do processo de prototipação e os requisitos fornecidos, a Figura 7 demonstra um exemplo do protótipo do estudo de caso:

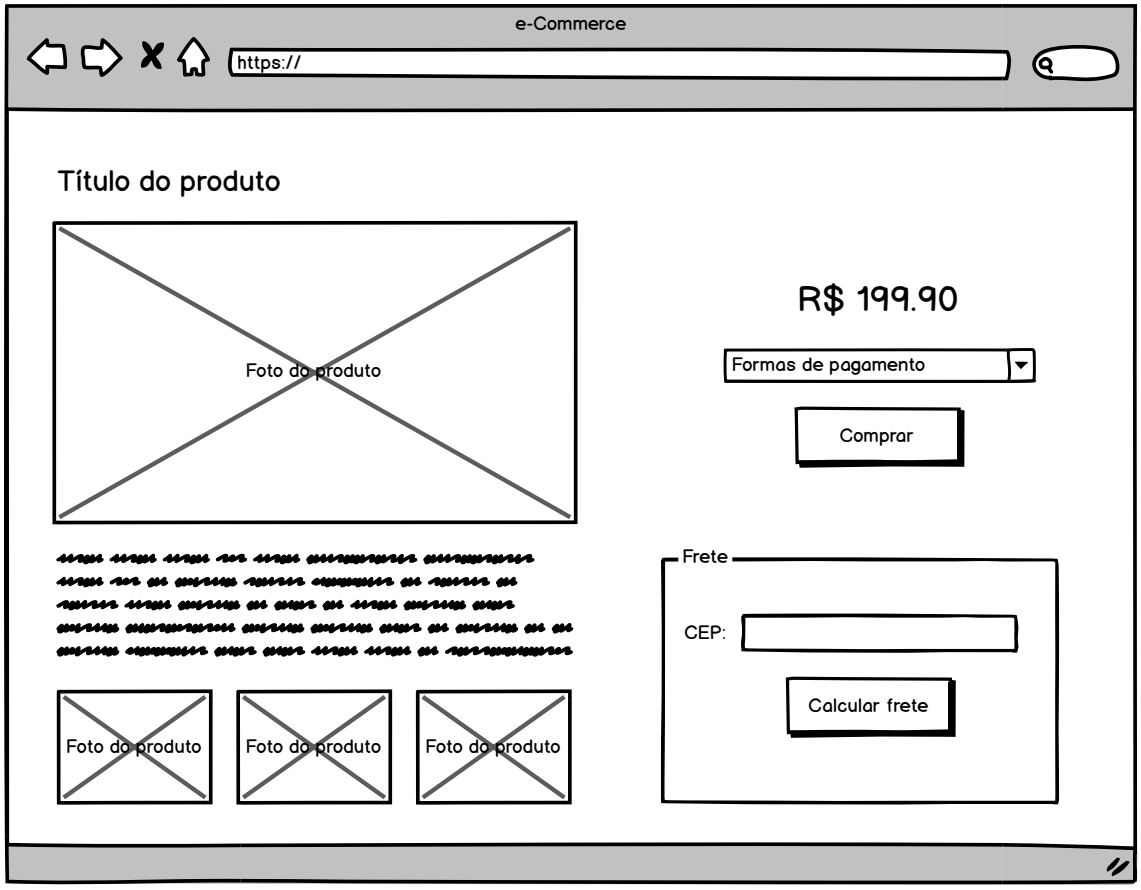

Figura 7. Protótipo da tela de especificação de produto

6.3. Especificar componentes na UI

Uma vez que os desenvolvedores estejam familiarizados com a UI (resultante do processo 1), a próxima fase consiste em relacionar a lista de funcionalidades com o protótipo criado. Isto é, identificar os domínios de negócio que representem cada fragmento da interface e quais são seus respectivos componentes de acordo com os requerimentos do

projeto, utilizando as mesmas diretrizes do processo do FDD. Essa classificação auxilia no agrupamento de elementos da interface de usuário e na posterior delegação de atividades de desenvolvimento, fatores que serão explorados nas próximas sub-seções.

Nesse processo, além da identificação dos componentes UI, os desenvolvedores podem estabelecer os contratos das *APIs*[2] a serem consumidas nos mesmos, a fim de definir as interfaces com os modelos de dados para facilitar a integração com o *back-end*.

Sendo assim, baseado no protótipo do estudo de caso, são observados três principais domínios: especificação do produto, compra do produto e frete. A Figura 8 mostra, com destaques coloridos, a divisão dos mesmos e quais são os respectivos componentes de UI:

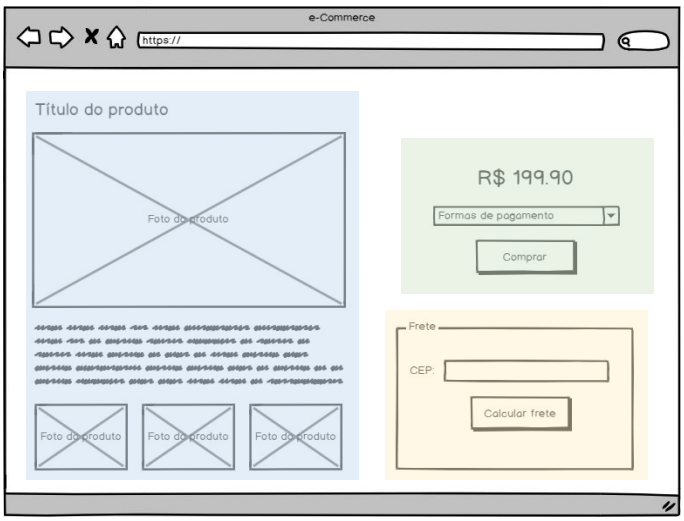

Figura 8. Protótipo da tela com seus domínios destacados

O final desse processo resulta nos três domínios de negócio da interface com seus respectivos componentes UI, assim como demonstrado na Tabela 2. Este mapeamento é utilizado pelos desenvolvedores pois fornece a especificação dos componentes que devem ser desenvolvidos em HTML.

Além disso, nessa fase é produzida a documentação referente aos contratos de APIs a serem desenvolvidas e posteriormente integradas. Para o estudo de caso, foram simuladas algumas APIs de acordo com o protótipo e os requerimentos, especificadas na lista a seguir:

1. API para informações de um produto
    - Descrição: Retorna informações do produto informado através do código identificador (id) do produto
    - URI: /produtos/idProduto
    - Método HTTP: GET
    - Exemplo de resposta da API:

[2]Provê um conjunto de protocolos e ferramentas para construir um software, realizando requisições remotas para obter informações através de protocolos padrões, como por exemplo o HTTP [Peters and McClennen 2016]

Domínio de negócio	Componentes UI
Especificação do produto	- Foto do produto - Rótulo com nome do produto - Parágrafo com descrição do produto - Carrossel com fotos do produto - Rótulo com o preço do produto
Compra do produto	- Caixa de seleção para formas de pagamento - Botão com rótulo "Comprar agora"
Frete	- Separador retangular com o rótulo "Frete" - Caixa de entrada numérica para CEP - Botão com rótulo "Calcular frete"

Tabela 2. Relação dos domínios de negócio da página e dos componentes UI


```
{
  "nome": "Exemplo de Nome do produto",
  "descricao": "Exemplo de Descrição do produto",
  "preco": 85.00,
  "imagens": {
    "principal": "url_main_image",
    "miniatura1": "url_small1_image",
    "miniatura2": "url_small2_image",
    "miniatura3": "url_small3_image"
  }
}
```


2. API para compra do produto
   - Descrição: API para envio de compra do produto a partir da opção escolhida pelo usuário
   - URI: /produtos/idProduto/comprar
   - Método HTTP: POST
   - Exemplo de requisição para envio da API:


```
{
  "opcaoCompra": 1,
  "idCliente": ""
}
```


3. API para cálculo de frete
   - Descrição: API para calcular o custo de entrega de um produto através do código postal fornecido pelo usuário
   - URI: /frete/codigoPostal
   - Método HTTP: GET
   - Exemplo de resposta da API:


```
{
  "endereco": "Rua Exemplo, 505",
  "bairro": "Bairro Exemplo",
  "cidade": "Cidade Exemplo",
  "estado": "SP",
  "codigoPostal": "89653155",
```

```
    "precoFrete" 58.3
}
```

6.3.1. *API First* e *Design de API*

Uma das boas práticas em projetos que utilizem *APIs* é o conceito de "*API First*"(ou, traduzido para o português, "API Primeiro"), que consiste em priorizar a definição dos contratos das *APIs* que serão utilizadas entre o *front-end* e o *back-end* em um projeto, ainda antes de iniciar a codificação de ambas as aplicações. Para tanto, é necessária a análise conjunta dos requisitos do projeto e do protótipo criado por ambos os times de desenvolvimento (*back-end* e *front-end*) [Biehl 2016]. Essa abordagem permite que, uma vez que os contratos estejam definidos, a aplicação consumidora da *API* não precise aguardar a conclusão do desenvolvimento da mesma, sendo que ambos os desenvolvimentos podem ser feitos paralelamente, economizando tempo e retrabalho [Rivero et al. 2013]. Em uma documentação de contratos de *API*, é necessário definir para cada uma a *URI* (trata-se de um identificador para qualquer fragmento de recurso online, incluindo URL [Liu et al. 2015]), quais serão os métodos do protocólo HTTP (*GET, POST, PUT, DELETE*, entre outros), quais serão os parâmetros utilizados nas requisições e como eles serão enviados (no corpo da requisição ou como parâmetro na URI) e qual será a resposta.

6.4. Planejar por componentes de UI

Após a identificação e documentação dos domínios de negócio e dos respectivos componentes UI, juntamente com os contratos das APIs já estabelecidas, o próximo passo é definir os desenvolvedores responsáveis pela implementação de cada domínio. Para tanto, o líder do projeto deve designar tais atividades para sua equipe, levando em consideração a prioridade e a interdependência de desenvolvimento de cada uma delas. Tais fatores dependem priorização dos clientes e das partes interessadas do projeto. No estudo de caso, a ordem de desenvolvimento dos componentes poderia ser Especificação do produto, Compra do produto e por fim Frete.

6.5. Projetar por grupos de UI

Após a atribuição das tarefas, é iniciado o processo de desenvolvimento dos componentes de interface de usuário, no qual as tecnologias, linguagens de programação e *frameworks* podem ser escolhidos a partir da preferência do desenvolvedor ou dos padrões utilizados no projeto. Neste momento, é desenvolvido o HTML extraído do protótipo criado e das especificações das atividades advindas dos processos Prototipar interface de usuário e Planjear por componentes UI, e então é realizada a integração com as *APIs* dos sistemas de *backend* caso seja aplicável.

Ao final do processo, todos os componentes de interface de usuário pertencentes ao domínio de negócio devem estar implementados e integrados com as APIs provedora de dados.

6.6. Construir por grupos de UI

O último processo trata-se de realizar testes nos componentes criados. Nos *frameworks* de *front-end* baseados em componentes, é comum a utilização de estruturas de testes unitários de código aberto, como o Jasmine e o Karma. Para testes de interface, o Selenium

pode ser utilizado como ferramenta para automatização bem como aplicar métodos de avaliação de interfaces de usuário, como a Avaliação Heurística[3]. Por fim, os testes de integração podem ser definidos entre os times de *back-end* e *front-end*.

Uma vez que os testes funcionem como o esperado, o domínio de negócio com seus respectivos componentes UI é apresentado ao cliente para a sua aprovação. Se aprovados, os componentes são, então, implantados juntamente com os demais já em funcionamento. Caso o cliente não aprove ou haja alterações necessárias antes da implantação, o desenvolvedor refaz o processo de Projetar por grupos de interface de usuário. Os dois últimos processos do FUiDD são iterativos, ou seja, são projetados para serem repetidos até que os requisitos envolvidos sejam atendidos.

Ao final das iterações, a interface de usuário deve estar desenvolvida. No estudo de caso, a Figura 9 representa o resultado da tela de visualização de produto:

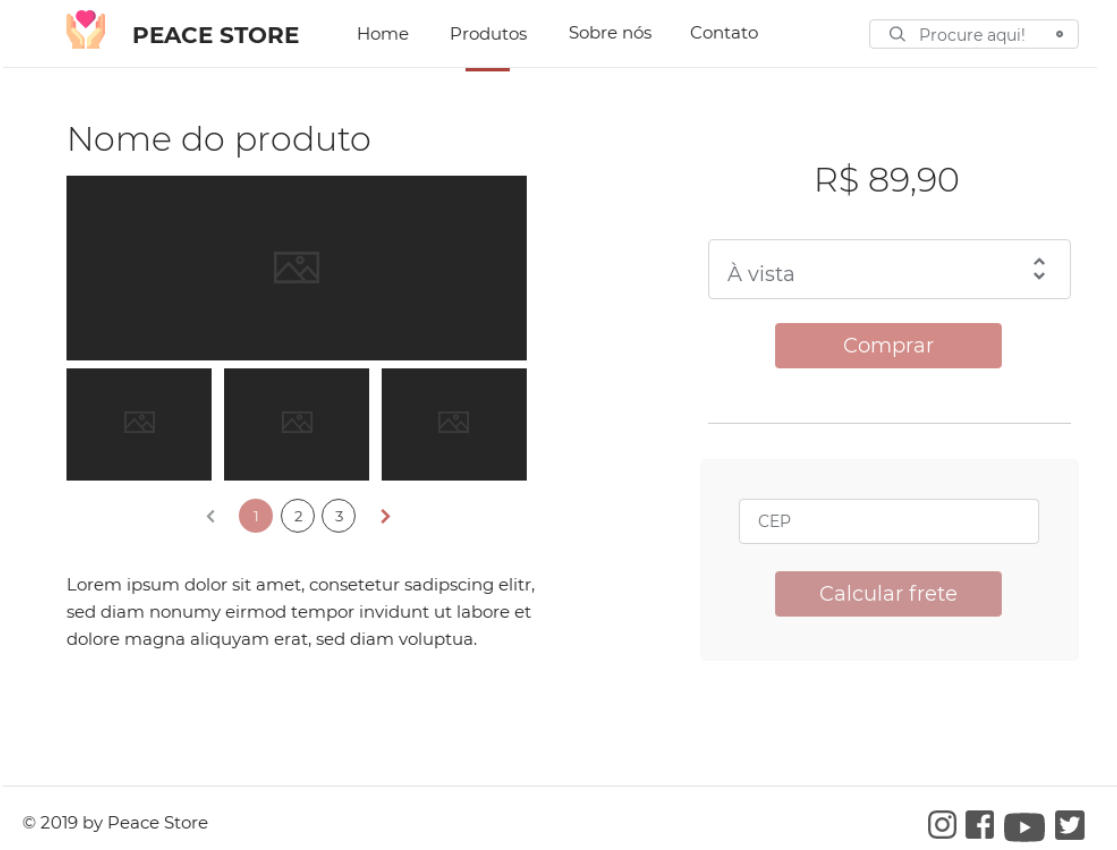

Figura 9. Resultado da interface de usuário após aplicar o FUiDD

7. Conclusão

Esta pesquisa foi desenvolvida dada a notória ausência de metodologias ágeis focadas no desenvolvimento de interfaces de usuário, sendo o objetivo do trabalho criar processos para a projeção, desenvolvimento, testes e implantação de interfaces de usuário. Para

[3]Método de análise de usabilidade em que 3 a 5 avaliadores são apresentados a um design de interface e os mesmos a avaliam de acordo com heurísticas propostas [Nielsen and Molich 1990]

tanto, foi estudado o *Feature Driven Development*, metodologia escolhida como referencial teórico, e analisados os processos comumente realizados ao desenvolver uma interface de usuário. Tais análises resultaram em cinco processos intitulados de *Feature User Interface Driven Development*, e, para testá-los, foi aplicado um estudo de caso de uma interface de usuário de uma aplicação de domínio do *e-Commerce*. Ao final do estudo, foi concluído que o objetivo foi atingido, assim como a contribuição para os estudos de metodologias de desenvolvimento para componentes de interface de usuário.

Para a realização deste trabalho, foram necessários conhecimentos aprofundados nas disciplinas de Interação Humano-Computador, Desenvolvimento de Sistemas Web, Arquitetura de Software, Engenharia de Software, Metodologias Ágeis, Gestão de Projetos, entre outras disciplinas cursadas.

8. Trabalhos Futuros

Os *frameworks* de *front-end*, assim como as tecnologias no geral, estão em constante evolução e, portanto, é esperado que novos métodos de desenvolvimento sejam criados. Logo, as metodologias de desenvolvimento devem se adequar afim de serem válidas diante de tal evolução. Sendo assim, é sugerido como trabalhos futuros a aplicação do *Feature User Interface Driven Development* em um novo estudo de caso, utilizando *frameworks* baseados em componentes mais novos daqueles já citados nesta pesquisa, focando na análise da eficiência do processo.

9. Agradecimentos

Por fim, para a conclusão deste trabalho só foi viável com a inestimável ajuda e o conhecimento do Professor Doutor André Constantino da Silva, a quem devo grande agradecimento e admiração. Ao professor Mestre Gustavo Bartz Guedes, por sua contribuição valiosa desde o início da pesquisa. À minha família, em especial meus avós, pela paciência e por nunca medirem esforços em ajudar a realizar meus sonhos. Sei que este é igualmente o sonho deles.

Referências

(2019). Angular vs react vs vue: npm trends. *npm trends: Compare NPM package downloads*.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., and Thomas, D. (2001). Manifesto for agile software development.

Biehl, M. (2016). *RESTful API design: APIs your consumers will love*, volume 3. API-University Press.

Firdaus, A., Ghani, I., and Jeong, S. R. (2014). Secure feature driven development (sfdd) model for secure software development. *Procedia - Social and Behavioral Sciences*, 129:546 – 553. 2nd International Conference on Innovation, Management and Technology Research.

Goyal, S. (2008). Major seminar on feature driven development. *Jennifer Schiller Chair of Applied Software Engineering*.

Harms, H., Rogowski, C., and Lo Iacono, L. (2017). Guidelines for adopting frontend architectures and patterns in microservices-based systems. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 902–907, New York, NY, USA. ACM.

Hasselbring, W. and Steinacker, G. (2017). Microservice architectures for scalability, agility and reliability in e-commerce. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 243–246.

Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., and Tilkov, S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35.

Liu, Z., Liu, H., Hardy, J., and Liu, L. (2015). Uniform resource identifier encoded by base62x. In *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*, pages 1087–1093.

Nielsen, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 249–256, New York, NY, USA. ACM.

Oh, J., Ahn, W. H., and Kim, T. (2017). Web app restructuring based on shadow doms to improve maintainability. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 118–122.

Palmer, S. R. and Felsing, M. (2001). *A Practical Guide to Feature-Driven Development*. Pearson Education, 1st edition.

Peters, S. E. and McClennen, M. (2016). The paleobiology database application programming interface. *Paleobiology*, 42(1):1–7.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Rivero, J. M., Heil, S., Grigera, J., Gaedke, M., and Rossi, G. (2013). Mockapi: An agile approach supporting api-first web application development. In Daniel, F., Dolog, P., and Li, Q., editors, *Web Engineering*, pages 7–21, Berlin, Heidelberg. Springer Berlin Heidelberg.

Shabib Aftab, Zahid Nawaz, F. A. M. S. B., Ahmad, M., and Anwar, M. (2018). Comparative analysis of fdd and sfdd. In *IJCSNS International Journal of Computer Science and Network Security*, volume 18.

Thakur, S. and Singh, H. (2014). Fdrd: Feature driven reuse development process model. In *2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies*, pages 1593–1598.

Thönes, J. (2015). Microservices. *IEEE Software*, 32(1):116–116.

Documento Digitalizado Público

Anexo I - artigo TCC

Assunto: Anexo I - artigo TCC
Assinado por: Andre Constantino
Tipo do Documento: Relatório Externo
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Original

Documento assinado eletronicamente por:

- **Andre Constantino da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 12/03/2020 15:52:38.

Este documento foi armazenado no SUAP em 12/03/2020. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 373141
Código de Autenticação: 9ad7765e3c