CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA DE FRANCA

“Dr. THOMAZ NOVELINO”

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GETÚLIO VINICIUS TEIXEIRA DA SILVA

DESENVOLVIMENTO DE SOFTWARE

Aspectos Gerais e Documentação

Franca/SP  
2018

GETÚLIO VINíCIUS TEIXEIRA DA SILVA

DESENVOLVIMENTO DE SOFTWARE

Aspectos Gerais e Documentação

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - “Dr. Thomaz Novelino”, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Dra. Jaqueline Brigladori Pugliesi

Franca/SP

2018

GETÚLIO VINíCIUS TEIXEIRA DA SILVA

DESENVOLVIMENTO DE SOFTWARE

Trabalho de Graduação apresentado à Faculdade de Tecnologia de Franca - “Dr. Thomaz Novelino”, como parte dos requisitos obrigatórios para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Trabalho avaliado e aprovado pela seguinte Banca Examinadora:

Orientador(a) : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Nome : Dr. Jaqueline Brigladori Pugliesi

Instituição : Faculdade de Tecnologia de Franca – “Dr. Thomaz Novelino”.

Examinador(a) 1 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Nome : Examinador\_1.

Instituição : Instituição\_1.

Examinador(a) 2 : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Nome : Examinador\_2.

Instituição : Instituição\_2.

Franca, \_\_ de junho de 2018.

AGRADECIMENTO

Xxxxxxxxxxxxx

Dedico o presente Trabalho de Graduação a Deus e aos meus familiares, em especial à minha esposa (Nome) e filhos (Nomes).

Aqui está uma pergunta: quando foi a última vez que você ouviu um argumento, e com base nesse argumento, mudou de ideia? Não apenas sobre algo que você realmente não pensou muito, mas algo que, antes de considerar o argumento em questão, sentiu-se bastante certo em relação à sua posição original.

Em outras palavras, quando foi a última vez que percebeu que estava completamente errado em uma questão de opinião?

Se a sua resposta é "nunca", ou mesmo "há muito tempo", é porque você está sempre certo?

John Gruber

RESUMO

O vídeo fornece uma maneira poderosa de ajudá-lo a provar seu argumento. Ao clicar em Vídeo Online, você pode colar o código de inserção do vídeo que deseja adicionar. Você também pode digitar uma palavra-chave para pesquisar online o vídeo mais adequado ao seu documento. Para dar ao documento uma aparência profissional, o Word fornece designs de cabeçalho, rodapé, folha de rosto e caixa de texto que se complementam entre si. Por exemplo, você pode adicionar uma folha de rosto, um cabeçalho e uma barra lateral correspondentes. Clique em Inserir e escolha os elementos desejados nas diferentes galerias. Temas e estilos também ajudam a manter seu documento coordenado.

**Palavras-chave:** Markdown. Documentação. Projeto de Software. Trabalho de Graduação.

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit amet commodo magna eros quis urna. Nunc viverra imperdiet enim. Fusce est. Vivamus a tellus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Proin pharetra nonummy pede. Mauris et orci. Aenean nec lorem. In porttitor. Donec laoreet nonummy augue. Suspendisse dui purus, scelerisque at, vulputate vitae, pretium mattis, nunc

**Keywords:** GPI. NBR 6.028. Pre-text. Summary. GPI Homework

LISTA DE FIGURAS

**Figura 1** – Xxxxxxxxxxxxx 10

**Figura 2** – Yyyyyyyyyyyyy 13

LISTA DE QUADROS

**Quadro 1** – Xxxxxxxxxxxxx 10

**Quadro 2** – Yyyyyyyyyyyyy 13

LISTA DE TABELAS

**Tabela 1** – Xxxxxxxxxxxxx 10

**Tabela 2** – Yyyyyyyyyyyyy 13

LISTA DE SIGLAS

**XXX** – Xxxxxxxxxxxxx

**YYY** – Yyyyyyyyyyyyyyyy

Sumário

[1 INTRODUÇÃO 15](#_Toc509157538)

[2 *KICK OFF* 17](#_Toc509157539)

[2.1 PROJETO DE *SOFTWARE* 18](#_Toc509157540)

[2.2 ENGENHARIA DE *SOFTWARe* 20](#_Toc509157541)

[2.2.1 Metodologias 21](#_Toc509157542)

[2.2.1.1 Métodos prescritivos 21](#_Toc509157543)

[2.2.1.2 Métodos ágeis 23](#_Toc509157544)

[2.2.1.3 Extreme Programming, Scrum e Kanban 24](#_Toc509157545)

[2.2.2 Ferramentas 27](#_Toc509157546)

[2.2.2.1 Linguagem natural e linguagem natural estruturada 28](#_Toc509157547)

[2.2.2.2 UML e BPMN 29](#_Toc509157548)

[2.2.2.3 Banco de dados 32](#_Toc509157549)

[2.2.2.4 Prototipação 34](#_Toc509157550)

[2.2.3 Qualidade de *software* 35](#_Toc509157551)

[3 DOCUMENTAÇÃO DE *SOFTWARE* 37](#_Toc509157552)

[3.1 DOCUMENTAÇÃO DO PROJETO DE *SOFTWARE* 37](#_Toc509157553)

[3.2 DOCUMENTAÇÃO DE *SOFTWARE* PARA USUÁRIOS 38](#_Toc509157554)

[3.2.1 Apresentação de funcionalidades e documentação operacional 39](#_Toc509157555)

[3.2.2 Documentação colaborativa 41](#_Toc509157556)

[3.2.3 Segurança da informação 42](#_Toc509157557)

[3.3 GERENCIAMENTO DA DOCUMENTAÇÃO 43](#_Toc509157558)

[4 LINGUAGEM MARKDOWN 44](#_Toc509157559)

[4.1 LINGUAGEM DE MARCAÇÃO LEVE 47](#_Toc509157560)

[4.2 ARQUIVOS *MARKDOWN* 47](#_Toc509157561)

[4.3 SINTAXE 47](#_Toc509157562)

[4.3.1 Parágrafos e linhas 49](#_Toc509157563)

[4.3.2 HTML incorporado 50](#_Toc509157564)

[4.3.3 Ênfase 51](#_Toc509157565)

[4.3.4 Cabeçalhos 53](#_Toc509157566)

[4.3.5 Listas 54](#_Toc509157567)

[4.3.6 Citações em bloco 58](#_Toc509157568)

[4.3.7 Links 60](#_Toc509157569)

[4.3.8 Imagens 63](#_Toc509157570)

[4.3.9 Código 64](#_Toc509157571)

[4.3.10 Caractere de escape 66](#_Toc509157572)

[4.4 *PARSERS* 67](#_Toc509157573)

[4.4.1 *Paser* criado por John Gruber 67](#_Toc509157574)

[4.4.2 Novos *parsers* 68](#_Toc509157575)

[4.4.3 Recursos implementados pelos novos *parsers* 71](#_Toc509157576)

[4.5 PADRONIZAÇÃO DAS ESPECIFICAÇÕES 73](#_Toc509157577)

[4.6 APLICAÇÃO DA LINGUAGEM MARKDOWN 74](#_Toc509157578)

[4.6.1 Aplicações de uso geral 74](#_Toc509157579)

[4.6.2 Geradores de sites estáticos 79](#_Toc509157580)

[5 APLICAÇÃO PRÁTICA - PROJETO 81](#_Toc509157581)

[5.1 CONCEPÇÃO 81](#_Toc509157582)

[5.2 PLANEJAMENTO 81](#_Toc509157583)

[5.2.1 Metodologia 81](#_Toc509157584)

[5.2.2 Ferramentas 81](#_Toc509157585)

[5.3 EXECUÇÃO 81](#_Toc509157586)

[5.4 FECHAMENTO 81](#_Toc509157587)

[CONCLUSÃO 82](#_Toc509157588)

[REFERÊNCIAS 83](#_Toc509157589)

1. INTRODUÇÃO

Seres humanos utilizam programas de computadores para realizarem tarefas que muitas vezes não são capazes de concluírem, ou não conseguiriam concluir em espaços de tempo satisfatórios com outras ferramentas.

O uso de papel e caneta, por exemplo, pode não ser suficiente para realizar de modo eficiente cálculos complexos de planejamento da produção de uma fábrica, ou ainda, de modo eficaz a identificação de padrões de som, de temperatura, entre outras atividades. Estes são problemas que necessitam, para melhor solução, do auxílio de ferramentas computadorizadas para o seu processamento.

Em seu curso de programação em linguagem *Python*, (GUANABARA, 2017) observa que em todas as áreas de atividade humana há um crescimento na demanda de programas para computadores e outros tipos de eletrônicos, bem como na procura por profissionais que atuam na área de Tecnologia da Informação (TI), especialmente programadores.

Exemplos...

Em resposta ao crescimento do setor, nota-se o surgimento de iniciativas como a organização sem fins lucrativos *Code.org*, que visa aumentar o número de programadores disponíveis no futuro, além da inclusão de mulheres e minorias sub-representadas no mercado de desenvolvimento de software (CODE.ORG, 2017).

A ação da *Code.org* é concentrada na educação básica, investindo em programas pedagógicos de iniciação a ciência da computação para crianças. O programa de iniciação tem sido adotado por escolas regulares, principalmente nos Estados Unidos e a iniciativa tem recebido investimento de empresas como *Google, Facebook e Microsoft* (CODE.ORG, 2017).

Tendo sido exposto alguns fatos acerca do cenário atual de desenvolvimento e de perspectivas de expansão deste mercado, em face as ações de organizações como a *Code.org*, pode-se concluir que existe uma dependência de *software* impactando diretamente a vida das pessoas na sociedade e diante dessa demanda surge a necessidade de empenhar esforços em estudos nessa área.

Neste cenário de TI, existe o ramo da Engenharia de *Software* (ES), uma área ampla de atuação que segundo (PRESSMAN, 2011, p. 29), é um processo que utiliza um conjunto de métodos e ferramentas que possibilitam o desenvolvimento de *software* com qualidade, e, segundo (SOMMERVILLE, 2011, p. 5), um dos motivos pelos quais a ES é importante refere-se ao fato de que “cada vez mais, indivíduos e sociedades dependem dos sistemas de *software* avançados”. Portanto, ainda segundo Sommerville “temos de ser capazes de produzir sistemas confiáveis econômica e rapidamente”.

Adentrando um pouco mais nessa área de ES, nota-se a existência de uma tarefa em específico que demanda especial atenção, a Documentação de *Software*. Portanto, a partir de noções fundamentais de ES que serão abordadas no capítulo 2 (Referencial Teórico), este Trabalho de Graduação (TG) tem como objetivo ser um caso de uso para o desenvolvimento de um tipo específico de software cuja documentação será construída com a linguagem *Markdown*.

Assim, o capítulo 2 trará, também, uma visão sobre os aspectos gerais do processo de desenvolvimento e os tipos de documentação dos quais um *software* demanda; o capítulo 3 terá uma introdução a linguagem *Markdown*; o capítulo 4 apresentará uma aplicação prática, ou seja, um projeto de *software* elaborado para que o seu processo de desenvolvimento seja documentado com o uso da linguagem *Markdown*; e por fim o capítulo final deste trabalho apresentará uma conclusão sobre o estudo em torno do processo de documentação de *software* e a aplicação da linguagem *Markdown.*

1. *KICK OFF*

A escolha deste tema para elaboração do TG foi motivada pela dificuldade em lidar com as tarefas impostas na disciplina de ES do curso de Análise e Desenvolvimento de Sistemas (ADS), especialmente a tarefa de documentação, na qual era necessária a apresentação de um projeto de *software* para acompanhamento dos estudos.

O marco inaugural foi a busca de uma alternativa ao modelo de Documento de Requisitos (DR) apresentado pela instituição, que consistia em um arquivo de texto, no formato “.docx”, contendo uma estrutura determinada para inclusão do conteúdo.

Ao término das atividades, foi obtida uma vasta documentação e não apenas um apanhado de requisitos, conforme o título levava a crer. Assim, o modelo finalizado continha informações oriundas de diversas ferramentas, forjadas através de inúmeras técnicas e, por vezes, abordagens distintas sobre um mesmo ponto do projeto.

Mais adiante no curso, conforme foram sendo apresentados novos conceitos da ES, somou-se ao DR a solicitação de tarefas que consistiam na elaboração de diagramas complementares e outros documentos textuais, em muitos aspectos redundantes.

Havia, portanto, uma grande confusão de técnicas, ferramentas e abordagens, bem como um processo pouco claro e generalista, que desencadeou em um grande desafio, qual seja, compreender e codificar tudo o que havia sido construído, em relação ao projeto apresentado, sob o prisma dos conceitos de ES.

Certamente havia a compreensão de quão necessário era, para o desenvolvimento de um projeto, a aplicação de um processo fundamentado em premissas da ES. Tal necessidade era fomentada pela ideia de construir, ainda na academia, um produto com finalidades comerciais, ou seja, usar todo o processo de ES, as ferramentas, os conceitos, enfim, qualquer boa prática existente para atribuir qualidade ao software que seria construído a partir do referido projeto.

Com um grande desafio já estipulado, e, considerando o conceito de “engenharia de *software* com o auxílio de computador” (PRESSMAN, 2011, p. 40), na qual ocorre a integração de ferramentas de modo a possibilitar que uma utilize informações geradas por outra, fez-se um novo desafio, o de unificar toda a documentação gerada no processo em um formato aceitável para publicação, de fácil manutenção e acessível, que por fim pudesse explicitamente indicar ao leitor que sua produção seguiu por caminhos almejados a partir do emprego da ES e suas metodologias.

Portanto, o que será visto a seguir é um estudo que a partir de conceitos e opiniões visa apresentar uma forma viável de promover a gestão da documentação produzida em um projeto de *software*.

* 1. PROJETO DE *SOFTWARE*

(PRESSMAN, 2011, p. 47) afirma que “todo projeto de *software* é motivado por alguma necessidade de negócio”, ou seja, uma demanda financeira, industrial ou comercial.

Não apenas na área de TI, mas em diversas áreas de atividade humana é possível observar o termo projeto, que é um jargão conhecido e comumente empregado no intuito de expressar a ideia de construir algo novo de forma organizada. Geralmente em um intervalo de tempo determinado.

Seguindo este princípio, em se tratando de programas de computador, geralmente o desenvolvimento se dá através de um projeto, que segundo (FILHO, 2003) conta com “uma data de início, uma data de fim, uma equipe de desenvolvimento (da qual faz parte um responsável, denominado ‘gerente de projeto’) e outros recursos”.

Ainda segundo (FILHO, 2003), um projeto representa a execução de um processo e este, quando bem definido, possui etapas elaboradas de modo a possibilitar uma avalição de progresso no desenvolvimento do projeto.

Assim, tem-se uma visão processual de projeto, ao passo que para (SOMMERVILLE, 2011, p. 25):

Um projeto de *software* é uma descrição da estrutura do *software* a ser implementado, dos modelos e estruturas de dados usados pelo sistema, das interfaces entre os componentes do sistema e, às vezes, dos algoritmos usados.

Tem-se, então, uma visão documental de projeto, e, pode-se concluir que ambas as visões, processual e documental, contemplam essencialmente a finalidade organizacional expressa por meio do jargão.

Deste modo, um processo generalista para um projeto de desenvolvimento de *software*, pode apresentar basicamente as etapas de:

1. Coleta de informações de entrada;
2. Execução de atividades que processam as informações da etapa 1; e
3. Construção do *software* com base nas especificações geradas na etapa 2.

Um modelo geral de processo de projeto, pode ser visto de forma abstrata na Figura 1 representando atividades comuns no desenvolvimento de um sistema de informação.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 1** - Um modelo geral do processo de projeto.

**Fonte:** (SOMMERVILLE, 2011, p. 26).

A conclusão de cada etapa envolve a execução de atividades distintas, que podem ser interligadas direta ou indiretamente. Além disso, as atividades a serem realizadas no processo variam de projeto para projeto, produzindo saídas em forma de documentos de especificações, quando se utiliza métodos prescritivos de desenvolvimento, ou, se a abordagem proposta for um método ágil de desenvolvimento, as saídas são representadas diretamente no código da aplicação (SOMMERVILLE, 2011, p. 26).

Metodologias serão apresentadas com mais detalhes na sessão 2.2.1, todavia, é possível notar, de modo antecipado, que a quantidade de documentação produzida possui uma relação direta de dependência com o tipo de projeto, o tamanho do projeto e a abordagem metodológica utilizada.

* 1. ENGENHARIA DE *SOFTWARe*

Em se tratando da origem e da serventia da ES, segundo o relato de (SOMMERVILLE, 2011, p. xi) “o nome ‘engenharia de *software’* foi proposto em 1969, na conferência da Organização do Tratado do Atlântico Norte (OTAN), para a discussão de problemas relacionados com desenvolvimento de *software*” e para (PRESSMAN, 2011, p. 49), “a prática da engenharia de *software* é uma atividade de resolução de problemas que segue um conjunto de princípios básicos”, ou seja, programas de computador, que devem existir para solução de problemas, possuem problemas.

Como consequência desta conclusão, pode-se constatar que, conforme citado na introdução deste TG, o objetivo da ES é a obtenção de *software* confiável dotado de qualidade. Em outras palavras, o tanto quanto possível livre de problemas.

A qualidade, na visão de (FILHO, 2003, p. 17), está estritamente relacionada com o grau de conformidade que o produto finalizado mantém com os respectivos requisitos, que “são as características que definem os critérios de aceitação de um produto” (FILHO, 2003, p. 13).

Isto conduz, naturalmente, ao princípio dos trabalhos em um projeto de *software*, que entre outras tarefas, primordialmente envolve o levantamento dos requisitos, talvez o primeiro artefato de documentação do projeto.

A documentação segue nas demais etapas do processo, que por sua vez é definido em conformidade com a visão de ES analogamente representada por camadas na Figura 2.

**Figura 2** - Camadas da Engenharia de *Software*.



**Fonte:** (PRESSMAN, 2011, p. 39).

Pode-se dizer que não existem variações no esquema proposto na ilustração, trata-se de:

* um objetivo o qual se pretende alcançar, o *software* de qualidade, e
* uma forma de se obter o que é pretendido, a sequência de um processo através dos seus componentes, os métodos e as ferramentas.

Partindo desta visão, sendo o projeto de desenvolvimento um empreendimento com data de início e data de conclusão previstas, chega-se ao o prazo para entrega, que é o primeiro problema do *software*, afinal, enquanto ele não fica pronto, o problema real que necessita de auxílio no processo de solução, segue sendo abordado de outras maneiras, através de outras ferramentas teoricamente menos eficientes.

O prazo, metaforicamente entendido como um problema a ser equacionado pela ES, é útil para ilustrar a aplicação do esquema de camadas, ou seja, tendo como foco a qualidade, define-se um processo para obtenção do produto baseado em fatores como por exemplo o prazo, entretanto o prazo não é o único fator a ser debatido ao se estabelecer um processo para o desenvolvimento do projeto. Outros fatores como o escopo do *software*, a aplicação, a quantidade de requisitos, a quantidade de usuários atendidos e a mão de obra disponível devem ser considerados para a adoção de métodos e ferramentas.

* + 1. Metodologias

As metodologias de desenvolvimento possuem uma diferenciação, de certo modo rasa, porém compreensível, entre os chamados métodos prescritivos e os métodos ágeis.

Sem adentrar no funcionamento e nas peculiaridades de cada um dos métodos existentes, o leitor deste TG pode considerar vago o propósito pelo qual este tópico foi abordado, contudo, é preciso salientar que a quantidade de documentos produzidos em um projeto de *software* depende do método empregado.

Assim, para melhor compreensão da proposta de documentação e suas aplicações, será feita uma breve introdução as metodologias de desenvolvimento mais frequentemente utilizadas.

* + - 1. Métodos prescritivos

Os métodos prescritivos sugiram entre as décadas de 1960 e 1970, na mesma época em que foi cunhado o termo Engenharia de *Software*. Tais métodos, segundo (ANTONIO, 2011), “previam alto nível de regulação no ciclo de vida do *software*, prescrevendo um arcabouço rígido de desenvolvimento e grande quantidade de documentação”.

Ainda segundo (ANTONIO, 2011), o surgimento dos métodos prescritivos se deu em função de um período anterior crítico denominado “Crise do *Software*”, o que coaduna com o relato de (SOMMERVILLE, 2011, p. xi) sobre os problemas no desenvolvimento de *software* abordados na OTAN.

Segundo (PRESSMAN, 2011, p. 60) o Modelo Cascata é um dos métodos prescritivos mais conhecidos, sendo o paradigma mais antigo da ES. Este método se aproxima em vários aspectos da atividade de projeto de *software* proposta pela instituição de ensino que foi citada no início deste capítulo.

Trata-se de uma abordagem que segue um processo estritamente linear começando pelo levantamento de requisitos junto ao cliente, seguindo pelo planejamento, pela modelagem, pela construção e, por fim, a implantação e o suporte ou manutenção do *software* (PRESSMAN, 2011, p. 59).

A Figura 3 ilustra as etapas do método denominado modelo cascata.

**Figura 3** - Modelo Cascata.

**Fonte:** (PRESSMAN, 2011, p. 60)

Em resumo, os métodos prescritivos são capazes de melhorar a qualidade do software, pois propiciam certas condições de gerenciamento que implicam em previsões realistas dos prazos, tendo em vista que se conhece as ferramentas e sabe-se exatamente quais documentos serão elaborados antes da codificação, quais etapas a codificação percorrerá e quais testes serão realizados, além de um procedimento para implantação.

Contudo, o processo que utiliza um método tal qual o modelo cascata enfrenta um problema crítico em relação ao seu caráter linear, pois uma etapa só é iniciada após a conclusão da anterior, de modo que ao detectar um erro na etapa de comunicação, estando o projeto na etapa de construção, por exemplo, será preciso refazer as etapas intermediárias de planejamento e modelagem, bem como a própria etapa de construção.

Por essa razão o modelo cascata não é o único método prescritivo que existe, pode-se elencar os modelos de Processo Incremental, Evolucionário e Espiral, que surgiram com a missão de amenizar este problema, além de outros modelos que não são ditos prescritivos totalmente, mas que possuem fortes ligações com os métodos prescritivos tradicionais, sendo eles os modelos de Processo Especializado, Processo Unificado, Processo Pessoal e de Equipe e o Processo de Produto.

No fim das contas, todos os métodos prescritivos, além de garantirem as condições de gerenciamento necessárias para a entrega de um dado projeto, são capazes de produzir grande quantidade de documentação.

* + - 1. Métodos ágeis

O Desenvolvimento Ágil, amplamente adotado a partir do lançamento do Manifesto Ágil (BECK, BEEDLE, *et al.*, 2001), permite empreender projetos de *software* dentro dos objetivos da ES para segmentos que possuem questões críticas como, por exemplo, mudanças de requisitos em função das necessidades de negócio, que por ventura venham ocorrer após o início do projeto.

Os autores do Manifesto Ágil fundamentaram sua publicação em dose princípios, onde: constata-se a intensão de valorizar a entrega do *software* no processo de desenvolvimento, fazendo-a de forma parcial e continua de modo a agregar valor (funcionalidades) em intervalos curtos; tem-se as entregas como parâmetro para aferir o progresso rumo a conclusão do projeto; preconiza-se as interações entre os envolvidos no processo favorecendo conversas face a face; e promove-se a sustentabilidade por meio de processos simplificados, tendo em vista a redução de trabalho, além de constantes avaliações sobre a eficácia, modificando a abordagem quando preciso (BECK, BEEDLE, *et al.*, 2001).

Isto de fato cria um contraponto ao modelo tradicional de desenvolvimento encabeçado pelos métodos prescritivos, onde: a implantação do *software* era realizada apenas no final do processo; o progresso era medido por conclusões de etapas, sendo que a saída de algumas etapas constituíam documentos de especificações; a documentação produzida também possuía o viés comunicativo para com os envolvidos; e cumpria-se fielmente o processo sem variações na abordagem em face a eficácia ou a não eficácia do seu emprego.

Este contraponto é perceptível no Manifesto Ágil, que diz:

Indivíduos e interações mais que processos e ferramentas

*Software* em funcionamento mais que documentação abrangente

Colaboração com o cliente mais que negociação de contratos

Responder a mudanças mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda (BECK, BEEDLE, *et al.*, 2001).

Contudo, segundo (PRESSMAN, 2011, p. 81), “o desenvolvimento ágil poderia ser mais bem denominado ‘engenharia de *software* flexível’”. Para ele, atividades básicas como comunicação, planejamento, modelagem, construção e emprego, vistas por exemplo no modelo cascata, podem ser notadas em métodos ágeis, onde ao invés de serem tidas como paradigmas constituem um conjunto mínimo de tarefas que impulsionam o desenvolvimento.

As atividades citadas invariavelmente fariam parte de um planejamento, porém, as origens dos projetos de desenvolvimento de *software*, que são as necessidades de negócio, passam constantemente por mudanças, e, segundo (SOMMERVILLE, 2011, p. 38), inicialmente os clientes (donos do negócio) consideram “impossível obter um conjunto completo de requisitos de *software* estável”.

Portanto, a flexibilidade está na capacidade de adaptar o projeto as novas necessidades na medida em que forem surgindo, ou seja, responder as mudanças em detrimento ao planejamento anterior, conforme previsto no manifesto.

* + - 1. Extreme Programming, Scrum e Kanban

O *Extreme Programming - XP* (em português, Programação Extrema), e o *Scrum* são duas metodologias consideradas expoentes do desenvolvimento ágil. No caso da XP, entre outras características, (PRESSMAN, 2011, p. 87) cita como valores da metodologia a intensão de evitar documentação volumosa como forma de comunicação e a restrição do desenvolvimento as necessidades imediatas, a fim de liberar as funcionalidades para uso rapidamente.

A Figura 4 exibe o processo adotado na metodologia *XP*, onde fica claro que as entregas de funcionalidades representam o progresso do projeto, conforme preconizado pelo manifesto ágil.

**Uma imagem contendo texto

Descrição gerada com muito alta confiançaFigura 4** - O processo da *Extreme Programming (XP)*.

**Fonte:** (PRESSMAN, 2011, p. 88)

Já no *Scrum*, a outra metodologia expoente do desenvolvimento ágil, segundo (SOMMERVILLE, 2011, p. 50), o processo é constituído de três fases: o planejamento, o *sprint* e o encerramento do projeto, conforme demonstrado na **Figura 5**.

Uma imagem contendo texto

Descrição gerada com alta confiança**Figura 5** - O processo *Scrum*.

**Fonte:** (SOMMERVILLE, 2011, p. 50).

O *sprint,* que é a segunda fase do processo, ocorre de forma recorrente a fim de obter as entregas incrementais de funcionalidades do *software*. Assim, o *Scrum,* que muitas vezes é tido como metodologia, e pode-se dizer que é um método, possui outra faceta pela qual é conhecido, a de ser um *framework* de desenvolvimento. Os próprios criadores do *Scrum* o definem com *framework* (SCHWBER e SUTHERLAND, 2017, p. 3).

A **Figura 6** demonstra quando é possível obter um *framework*, a partir da comparação entre três atividades distintas A, B e C.

**Figura 6** - Criação de um framework.

**Fonte:** (Frameworks) online.

Um conjunto de soluções constitui um *framework*, que pode ser utilizado em várias aplicações, ou seja, soluciona problemas que são inerentes ao desenvolvimento de determinadas atividades, ainda que essas atividades não sejam relacionadas diretamente ou pertençam a áreas distintas do conhecimento.

Voltando ao *Scrum*, durante a fase do *sprint* surge a oportunidade de implementar outras metodologias no processo, como por exemplo o *Kanban*.

É comum a adoção do *Scrum* em conjunto com o *Kanban* que possui uma dinâmica de fácil compreensão, consistindo basicamente em um quadro colunado onde: cada coluna representa um estado para as tarefas; as tarefas são representadas por cartões contendo as informações básicas daquilo que deve ser executado; os cartões se movem da esquerda para a direita no quadro representando o progresso da execução de uma tarefa.

Comumente, em um quadro *Kanban,* tem-se as seguintes colunas intituladas: tarefas para fazer, tarefas em andamento e tarefas concluídas, conforme demonstrado na **Figura 7** (em inglês *TO DO, WORK IN PROGRESS* e *DONE*, respectivamente).

**Figura 7** - Exemplo de *Kanban* com tarefas.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** (MARTINS) online.

Eventualmente a quantidade de listas no quadro pode variar de acordo com as necessidades de um projeto, incluindo dessa forma novos estados para as tarefas ou até mesmo substituindo os que foram apresentados.

Uma das características do *Kanban* remete ao princípio de priorização das tarefas e, no quadro apresentado na Figura 7, pode-se considerar que foi adotado um método de priorização baseado nas cores dos cartões. Um cartão da cor vermelha deve ser iniciado antes de um cartão amarelo por exemplo. Este tipo de convenção é decidido na primeira fase do *Scrum,* caso tenha sido essa a metodologia adotada.

De modo geral, em métodos ágeis os requisitos levantados no início do projeto são desenvolvidos de forma incremental, obedecendo as prioridades estabelecidas pelos usuários (clientes).

* + 1. Ferramentas

Algumas ferramentas podem ser mais bem utilizadas por uma ou outra metodologia. O emprego de determinadas ferramentas implica na análise em particular do projeto que se deseja empreender, bem como a metodologia a ser adotada.

Inicialmente, tem-se a tarefa de levantamento dos requisitos, os quais o produto deverá estar em conformidade quando o projeto estiver concluído. Para essa tarefa em específico, existe um amplo estudo denominado Engenharia de Requisitos, que dispõe de alguns mecanismos que auxiliam a coleta de tais dados.

Segundo (o professor Ely Prado), em sua aula sobre levantamento de requisitos, algumas das ferramentas utilizadas, também chamadas de técnicas, são: entrevistas, estórias de usuários, etnografia que consiste na observação das tarefas durante a execução, casos de usos e outras, sendo que, após coletados os requisitos devem ser documentados.

A documentação pode se dar de três formas:

* Texto com linguagem natural ou linguagem natural estruturada;
* Modelo conceitual; ou
* Híbrido, contemplando as duas formas anteriores.
  + - 1. Linguagem natural e linguagem natural estruturada

O documento elaborado com linguagem natural possibilita o entendimento dos requisitos por todos os envolvidos no projeto, mesmo aqueles que não são especializados no desenvolvimento de *software*, sendo necessário apenas a compreensão do conjunto de regras de negócio.

O Quadro 1 exibe um trecho de um documento de requisitos no qual é possível observar o uso de linguagem natural para descrever os requisitos de um *software.*

**Quadro 1** - Exemplo de requisitos para o sistema de *software* de bomba de insulina.

|  |
| --- |
| Requisitos em linguagem natural |
| ...  3.2 - O sistema deve medir o açúcar no sangue e fornecer insulina, se necessário, a cada dez minutos. (Mudanças de açúcar no sangue são relativamente lentas, portanto, medições mais frequentes são desnecessárias; medições menos frequentes podem levar a níveis de açúcar  desnecessariamente elevados.)  ...  3.6 - O sistema deve, a cada minuto, executar uma rotina de autoteste com as condições a serem testadas e as ações associadas definidas no Quadro 4.3 (A rotina de autoteste pode descobrir problemas de hardware e software e pode alertar o usuário para a impossibilidade de operar  normalmente.)  ... |

**Fonte:** (SOMMERVILLE, 2011, p. 67).

No Quadro 1 é possível notar, no requisito identificado como item 3.6, que há uma referência a um trecho do documento onde os requisitos são documentados com linguagem natural estruturada. Este trecho é visto no Quadro 2.

**Quadro 2** - Uma especificação estruturada de um requisito para uma bomba de insulina.

|  |  |
| --- | --- |
| Requisitos em linguagem natural estruturada | |
| Bomba de insulina/Software de controle/SRS/3.3.2 | |
| Função | Calcula doses de insulina: nível seguro de açúcar. |
| Descrição | Calcula a dose de insulina a ser fornecida quando o nível de açúcar está na zona de segurança entre três e sete unidades. |
| Entradas | Leitura atual de açúcar (r2), duas leituras anteriores (r0 e r1). |
| Fonte | Leitura atual da taxa de açúcar pelo sensor. Outras leituras da memória. |
| Saídas | CompDose - a dose de insulina a ser fornecida. |
| Destino | Loop principal de controle. |
| Ação | CompDose é zero se o nível de açúcar está estável ou em queda ou se o nível está aumentando, mas a taxa de aumento está diminuindo. Se o nível está aumentando e a taxa de aumento está aumentando, então CompDose é calculado dividindo-se a diferença entre o nível atual de açúcar e o nível anterior por quatro e arredondando-se o resultado. Se o resultado é arredondado para zero, então CompDose é definida como a dose mínima que pode ser fornecida. |
| Requisitos | Duas leituras anteriores, de modo que a taxa de variação do nível de açúcar pode ser calculada. |
| Pré-condição | O reservatório de insulina contém, no mínimo, o máximo de dose única permitida de insulina. |
| Pós-condições | r0 é substituída por r1 e r1 é substituída por r2. |
| Efeitos colaterais | Nenhum. |

**Fonte:** (SOMMERVILLE, 2011, p. 68).

* + - 1. UML e BPMN

A documentação elaborada a partir de modelos conceituais implica em prévio conhecimento da notação utilizada. Por exemplo, em projetos desenvolvidos com base no paradigma da Orientação a Objeto, é comum o uso de diagramas da UML - *Unified Modeling Language* (em português, Linguagem de Modelagem Unificada). A Figura 8 exibe um modelo conceitual como um diagrama UML de caso de uso para um sistema bancário.

**Figura 8** - Exemplo do sistema bancário com caso de uso e atores.

**Uma imagem contendo texto, mapa

Descrição gerada com muito alta confiançaFonte:** (OBJECT MANAGEMENT GROUP, 2017, p. 643)

As figuras de bonecos de palito do diagrama representam os atores, que são os responsáveis por executar as ações, e estas são representadas pelas figuras elípticas contendo os verbos, referentes as ações, inscritos em seus interiores (OBJECT MANAGEMENT GROUP, 2017, p. 644 e 645).

A UML possui diversos diagramas capazes de exemplificar de acordo com o seu propósito os requisitos levantados, sendo confirmada, portanto, a necessidade de o indivíduo envolvido com o projeto conhecer a notação empregada para compreender os requisitos documentados com um modelo conceitual.

Neste ponto, observa-se a ligação entre a tarefa de levantamento de requisitos e a consequente tarefa de modelagem do *software*. Esta, por sua vez, além de contar com a linguagem UML, possui outras ferramentas como, por exemplo, a modelagem de processos denominada *Business Process Model and Notation* (BPMN), em português, Notação de Modelagem de Processos de Negócio.

A modelagem BPMN consiste na elaboração de diagramas e foi desenvolvida para representar graficamente os processos de um negócio, não se atendo apenas ao desenvolvimento de *software*.

O principal objetivo do BPMN é fornecer uma notação que seja facilmente compreensível por todos os usuários empresariais, do negócio: os analistas que criam os rascunhos iniciais dos processos, os desenvolvedores técnicos responsáveis pela implementação da tecnologia que irá realizar esses processos e, finalmente, os empresários que gerenciarão e monitorarão aqueles processos (OBJECT MANAGEMENT GROUP, INC. (OMG), 2011, p. 1).

Deste modo, os diagramas BPMN são constantemente utilizados durante o processo de desenvolvimento de *software*, justamente por sua contribuição considerável para o entendimento e a elaboração dos processos os quais um *software* deverá servir como ferramenta de apoio.

Um diagrama BPMN possui elementos gráficos (figuras geométricas) que são dispostas de modo a estabelecer um fluxo do princípio ao fim de um processo, inclusive abrangendo subprocessos.

Nos diagramas é possível observar a interação entre atores de um processo. Por exemplo, os setores de uma empresa que são responsáveis pela realização de tarefas que se complementam mutuamente e constituem o processo, são identificados como atores do processo.

Assim, no diagrama BPMN as tarefas pertinentes a cada ator são dispostas em raias especificas, pois, analogamente, vê-se o processo como uma piscina olímpica e as raias dessa piscina representam os atores.

A Figura 9, traz um exemplo de diagrama BPMN simulando a comunicação entre um cliente e um fornecedor no processo de expedição de um cartão de crédito.

**Figura 9** - Fluxo de mensagens que se conecta a objetos de fluxo em duas piscinas.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** (OBJECT MANAGEMENT GROUP, INC. (OMG), 2011, p. 113)

No exemplo, é possível observar que a piscina inferior, que representa a parcela do processo executada pelo Fornecedor (*Supplier*), possui duas raias, atores, representando os setores de Vendas (*Sales*) e Distribuição (*Distribuition*) e a piscina superior, que representa a parcela do processo executada pela Instituição Financeira (*Financial Institution*) possui apenas uma raia, indicando que toda da instituição representa um ator no processo.

* + - 1. Banco de dados

Outro tipo de modelagem, este específico para projetos de *software* que utilizam Banco de Dados (BD), é o Modelo Entidade Relacionamento (MER). A elaboração desse modelo, parte de uma concepção denominada Modelo Conceitual, onde o BD “é descrito de forma independente de implementação” (HEUSER, 2009, p. 25).

A abordagem Entidade Relacionamento (ER), de onde deriva o MER, consiste em uma técnica de representação do BD através de um diagrama denominado Diagrama Entidade Relacionamento (DER), ilustrado pela Figura 10.

**Uma imagem contendo texto

Descrição gerada com muito alta confiançaFigura 10** - Diagrama Entidade Relacionamento (DER) para sistema de farmácias.

**Fonte:** (HEUSER, 2009, p. 67).

O diagrama apresentado, elaborado para um sistema de farmácias, permite observar:

* as entidades que são representadas por retângulos com o nome de cada entidade inscrito no interior do retângulo;
* os relacionamentos entre as entidades representados por losangos;
* as entidades associativas, que são representadas por losangos inscritos em retângulos;
* a indicação de especialização de uma entidade representada pelo triangulo;
* as respectivas conexões entre os elementos dadas pelas linhas; e
* os números entre parênteses, que representam a cardinalidade, ou seja, tipo de relacionamento entre duas entidades.

Quanto a cardinalidade, utilizando como exemplo o relacionamento entre LOTE e PRODUTO, demonstrado no diagrama da Figura 10, diz-se que, um LOTE possui no mínimo um e no máximo *n* PRODUTOS, ao passo que um PRODUTO pode pertencer no mínimo a nenhum LOTE e no máximo a *n* LOTES.

Outra concepção relacionada com a elaboração de BD é o Modelo Lógico, o qual é constituído por uma descrição textual e estruturada das tabelas de um BD.

O Modelo Lógico representa uma abstração do BD e é tido como uma transformação do Modelo Conceitual a fim de implantá-lo em um determinado Sistema de Gerenciamento de Banco de Dados (SGDB).

Neste ponto da modelagem tem-se uma estrutura representativa com todas as tabelas e os respectivos atributos (nomes das colunas em uma tabela do BD) conforme mostrado no Quadro 3.

**Quadro 3** - Modelo Conceitual de um cadastro de produtos.

|  |
| --- |
| Modelo Conceitual - Cadastro de Produtos |
| TipoDeProduto (CodTipoProd, DescrTipoProd)  Produto (CodProd, DescrProd, PrecoProd, CodTipoProd)  CodTipoProd referencia TipoProduto |

**Fonte:** (HEUSER, 2009, p. 27)**.**

O trabalho de modelagem do BD implica, além do levantamento de requisitos, no entendimento sobre as regras de negócio as quais o *software* estará sujeito, pois são elas que determinam os tipos de dados e as especificações as quais os dados deverão estar em conformidade antes de serem armazenados.

As especificações, ou seja, a tipificação dos dados aparece em outros tipos de representação do BD e, segundo (HEUSER, 2009, p. 24), “cada representação de um modelo de dados através de uma linguagem de modelagem recebe a denominação de Esquema de Banco de dados”.

* + - 1. Prototipação

O termo protótipo remete a ideia de primeiro exemplar de um produto, uma versão preliminar, o qual é utilizado para fins de teste e aperfeiçoamento. Em relação ao desenvolvimento de *software*, a tarefa de prototipação “tem como objetivo principal validar os requisitos, abordar questões de interface, e avaliar tanto a viabilidade quanto a complexidade do sistema”, segundo (SOUZA, VALE e ARAÚJO, 2008, p. 45).

Assim, esperasse com um protótipo expor os conceitos do projeto, sendo, portanto, uma prática comum que pode tanto ser desenvolvida com o auxílio de programas de computador quando através de desenhos feitos a mão.

Pode-se classificar os protótipos em três categorias: baixa fidelidade, média fidelidade e alta fidelidade. Deste modo, os protótipos de baixa fidelidade são destinados a crítica do produto em relação ao levantamento de requisitos; os protótipos de média fidelidade são destinados a representação visual do produto de maneira fidedigna; e por fim os protótipos de alta fidelidade destina-se ao teste componentes e solução de problemas técnicos.

Por conseguinte, percebe-se que a medida que a fidelidade de um protótipo avança da baixa para a alta, o esforço, o tempo gasto e consequentemente o custo de produção tende a aumentar. Deste modo, o uso dessa ferramenta é regrado por metodologias ágeis que preconizam a avaliação do custo benefício que uma tarefa desta natureza possui, ou seja, se o valor agregado pela adoção de protótipos de todas as categorias compensa o esforço, o tempo e o custo de desenvolvimento da tarefa.

No entanto, considerando a escolha do nível de fidelidade dos protótipos de um projeto algo estritamente ligado ao seu tamanho e relevância, um ponto de partida interessante, por exemplo, para projetos de aplicações *web*, é o desenvolvimento de *wireframes*, que permitem vislumbrar o produto e criticar questões de interface e experiência de usuário em um nível de fidelidade aceitável para grande parte dos projetos empreendidos por desenvolvedores individualmente ou em pequenas empresas com recursos limitados.

A Figura 11 apresenta um exemplo de *wireframe* denominado *mockup*, representando a tela de autenticação em um sistema *web*.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 11** - Protótipo de tela de *login* com *mockup*.

**Fonte:** (MALHERBI, 2013)*online*.

* + 1. Qualidade de *software*

Enfim a almejada qualidade de *software* vem à tona trazendo consigo uma série de questões subjetivas que dividem opiniões e motiva o estudo acerca do processo de desenvolvimento. O tema qualidade já foi abordado no início da sessão 2.2, de modo este trecho pretende apenas completar o raciocínio iniciado anteriormente.

Inicialmente, retomando a definição de qualidade, (SENE, 2012, p. 46) recorre a NBR (norma brasileira) 13.596 de agosto de 1996, que descreve qualidade como “a totalidade de características de um produto de *software* que lhe confere a capacidade de satisfazer necessidades explícitas e implícitas”. Neste ponto, ainda segundo (SENE, 2012, p. 47) a “qualidade é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos”.

Deste modo, nota-se que a qualidade depende de inúmeros fatores e que não se trata de algo trivial, afinal, satisfazer necessidade explicitas e implícitas utilizando um processo sistemático que previne e elimina defeitos é uma atividade deveras complexa.

Assim, limitando esta pesquisa ao foco deste TG, que é a documentação de *software*, um dos fatores que implicam em qualidade, e, abordando o tema sob essa ótica, é possível retomar um ponto já discutido que trata da importância dada a tarefa de documentação por diferentes metodologias de desenvolvimento, observando que, em relação a métodos:

Algumas pessoas pensam que a qualidade de *software* pode ser alcançada por meio de processos prescritivos, baseados em padrões organizacionais e procedimentos de qualidade associados que verificam que esses padrões serão seguidos pela equipe de desenvolvimento de *software*. Seu argumento é que os padrões incorporam as boas práticas de engenharia de *software* e que segui-las levará a produtos de alta qualidade. Na prática, contudo, existe muito mais no gerenciamento de qualidade do que apenas padrões e a burocracia associada para garantir que sejam seguidos (SOMMERVILLE, 2011, p. 456).

Talvez essa visão colabore para a adoção de metodologias ágeis em detrimento aos métodos prescritivos, consequentemente tem-se a diminuição da tarefa de documentação.

Ainda assim, existem aqueles que defendem e priorizam o processo de documentação e análise detalhados, enquanto outros, em razão do que foi exposto, consideram que manter o foco no código e entregar funcionalidades o mais rapidamente possível seja o ponto mais importante a se observar no desenvolvimento.

Além disso, nota-se que ao adotar métodos ágeis de desenvolvimento, às vezes, ao concluir a elaboração de um documento de requisitos, este já está ultrapassado e consequentemente o trabalho empenhado foi perdido.

Nessa peleja, não há que se dizer que um lado tem razão em detrimento ao outro. Cabe apenas observar que um repositório de informações, ou seja, de documentação do projeto, é algo conveniente a todos aqueles que em dado momento necessitam se orientar em relação ao trabalho que deve ser desenvolvido.

Por fim, a qualidade do *software* não se define apenas durante o desenvolvimento. Há que se pensar na qualidade do *software* considerando todas as suas partes, inclusive a documentação. Deste modo, é preciso observar que o usuário deve dispor de manuais de utilização que possam conduzi-lo a correta operação do *software*.

1. DOCUMENTAÇÃO DE *SOFTWARE*

Considerando o *software* como produto, ou seja, resultado de um processo, pode-se definir documentação de um *software* como sendo todo material elaborado com a finalidade de tornar o produto compreensível para desenvolvedores, engenheiros, usuários, representantes comercias, enfim, qualquer indivíduo que em determinado momento necessite de informações do projeto, o que pode ocorrer durante os processos de planejamento e desenvolvimento ou quando o *software* já se encontra em fase de produção, ou seja, em uso.

Muitas pessoas pensam que *software* é simplesmente outra palavra para programas de computador. No entanto, quando falamos de engenharia de *software*, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente. Um sistema de *software* desenvolvido profissionalmente é, com frequência, mais do que apenas um programa; ele normalmente consiste em uma série de programas separados e arquivos de configuração que são usados para configurar esses programas. Isso pode incluir documentação do sistema, que descreve a sua estrutura; documentação do usuário, que explica como usar o sistema; e sites, para usuários baixarem a informação recente do produto (SOMMERVILLE, 2011, p. 3).

O propósito de se documentar um *software* atende a duas necessidades, quais sejam: facilitar a comunicação entre eventuais membros da equipe ou entre a equipe e o cliente; e facilitar o entendimento das funções em eventuais atividades de manutenção ou atualização do *software* (FURTADO, 2016).

A abordagem de (SOMMERVILLE, 2011, p. 3), extraida de uma obra voltada para área de ES, faz menção tanto a documentação estrutural de um *software* quanto a documentação para o usuário final. Nota-se ali importância que é dada a documentação, sob todos os aspectos, na medida que foi considerada como parte do *software* e não algo que o acompanha em sua distribuição. Em outras palavras, a documentação faz parte daquilo que define o *software*.

A mesma visão pode ser observada na definição de (PRESSMAN, 2011, p. 32):

*Software* consiste em: (1) instruções (programas de computador) que, quando executadas, fornecem características, funções e desempenho desejados; (2) estrutura de dados que possibilitam aos programadores manipular informações adequadamente; e (3) informação descritiva, tanto na forma impressa como na virtual, descrevendo a operação e o uso de programas.

* 1. DOCUMENTAÇÃO DO PROJETO DE *SOFTWARE*

Durante o processo de desenvolvimento *software*, caso a equipe ou mesmo alguém que trabalha individualmente não anote, não documente aquilo que está fazendo sob vários aspectos como por exemplo os objetivos, as ações tomadas, mas principalmente para que serve e como deve ser utilizado cada um dos componentes do *software*, seguramente em um dado momento haverá, entre outros problemas alguns questionamentos como:

* Para que serve e por que tal componente foi desenvolvido?
* Como usar?
* Quais as dependências?
* Quem fez?

Esses questionamentos devem obter respostas para que o projeto prossiga e possa ser entregue. Portanto, aceitando-se as definições de *software* apresentadas anteriormente e considerando o que já foi abordado acerca de projeto e engenharia de software, constata-se que, durante o processo devem ser elaborados muitos diagramas, protótipos, especificações, instruções e outras tentativas de esclarecer as ideias, diretrizes, procedimentos, normas e afins.

Assim, todo este conteúdo produzido é denominado documentação de projeto de *software* e esse trabalho é explicado por (FURTADO, 2016) conforme segue:

Das definições de engenharia de software, a mais adequada para a documentação de projetos é que ela é uma área da computação voltada para a especificação, desenvolvimento e manutenção de sistemas de software, com aplicação de tecnologias e práticas de gerência de projetos e outras disciplinas visando organização, produtividade e qualidade.

Deste modo, tem-se novamente o foco na qualidade do produto como objetivo. Nesse sentido, algo que deve ser obtido ao produzir a documentação do projeto, seja através de uma plataforma de modelagem com notação específica ou através de textos em linguagem natural ou natural estruturada, é a definição precisa do requisito, ou seja, uma especificação de características de funcionamento que seja suficiente para codificar, testar e implementar o *software* (Ely Prado).

* 1. DOCUMENTAÇÃO DE *SOFTWARE* PARA USUÁRIOS

Esta parte da documentação do *software*, voltada para o usuário, descreve os modos de instalação, parametrização e operação.

* + 1. Apresentação de funcionalidades e documentação operacional

Algumas empresas desenvolvedoras, ao distribuírem os *softwares* optam por oferecer algum tipo de manual impresso, contendo a descrição das funcionalidades e modos de operação. Porém, a prática mais comum neste sentido é a distribuição do manual operacional do *software*, e até mesmo material de apresentação das funcionalidades, em formato digital.

Outra prática comum é a distribuição dos manuais operacionais em formato de páginas de hipertexto estáticas, as páginas HTML[[1]](#footnote-1) *- Hiper Text Markup Language* (em português, Linguagem de Marcação para Hipertexto), que são executadas localmente no computador do usuário, ou então em arquivos no formato PDF - *Portable Document File* (em português pode-se considerar Formato de Documento Portátil).

Entretanto, nota-se as opções de ajuda, que são encontradas a partir de algum atalho na interface do *software* (neste caso considera-se interface gráfica), atualmente tem encaminhado o usuário diretamente para a página de ajuda *on-line* no site da empresa desenvolvedora*.*

Evidentemente este modo distribuição torna mais simples a manutenção da documentação para usuários finais do *software*, considerando principalmente os processos de atualizações, onde é comum o *software* receber novas funcionalidades. Assim, ao atualizar o software o usuário o pacote de atualização não é distribuído com a documentação nos formatos citados anteriormente.

Também existem softwares criados especialmente para a leitura de documentação, como o caso do *Man*, bastante utilizado em terminais de Interface de Linha de Comando (CLI) dos Sistemas Operacionais (SO)s *GNU/Linux*. Neste caso em especial, a documentação de um dado *software* é formatada em arquivo estruturado de forma específica e empacotada juntamente com os binários de instalação do programa. Durante a instalação do software ocorre a indexação da documentação na base de dados do programa *Man*, tornando-a acessível para o usuário do computador.

Um exemplo de utilização do *Man* pode ser observado na Figura 12, onde é solicitada a documentação do software de controle de versão *Git* através da CLI em um terminal do SO *Debian GNU/Linux*.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 12** - Chamada ao *Man* para exibição da documentação do *git*.

**Fonte:** autoria própria.

A Figura 13 exibe no terminal do SO *Debian GNU/Linux* um trecho da documentação do *Git* apresentada pelo *Man.*

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 13** - Documentação do *Git* exibida através do *Man*.

**Fonte:** autoria própria.

Ademais, quanto a interação do usuário com o programa de computador, sabe-se que existem *softwares* de vários tipos. Tomando como exemplo um tipo especial de *software*, os chamados *frameworks*, que são voltados para desenvolvedores de *software*, e sua definição é a mesma adotada na sessão 2.2.1.3, especificamente demonstrada na Figura 6, constata-se o quão importante é a tarefa de documentação.

A Figura 14 exibe um trecho da documentação de um *framework* para desenvolvimento de aplicações *web* chamado *Vue.js*, utilizado para construção de componentes em linguagem *Javascript*, amplamente utilizada hodiernamente.

**Figura 14** - Trecho da documentação do *framework Vue.js*.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** (VUE.JS, 2018) online.

A documentação desse tipo de *software* voltada para o usuário, é imprescindível para sua utilização e a qualidade da mesma pode refletir no sucesso ou o fracasso do *software* em relação a sua adoção pelo mercado, afinal, partindo do exemplo exposto, sem a demonstração da maneira como se utiliza o *software,* não haveria nele serventia alguma.

* + 1. Documentação colaborativa

Outro ponto no qual a documentação do *software* para o usuário possui destaque, é no tocante a grandes projetos de *software* livre (*free software*) e *software* de código aberto(*open source*).

Nestes casos, forma-se uma comunidade de desenvolvedores que compartilham além de conhecimento e código, tempo e trabalho na produção de guias de usuário, em especial os manuais introdutórios e tutoriais para quem deseja aprender a usar o software.

Um exemplo desse trabalho comunitário é o *software* livre para desenho vetorial *Inkscape*. A Figura 15 exibe um trecho de uma página no site do projeto que mantem o desenvolvimento do software, no qual é possível observar links para conteúdo produzido pela comunidade.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 15** - Links para documentação colaborativa do sofware Inkscape.

**Fonte:** (INKSCAPE, 2018) online.

* + 1. Segurança da informação

Considerando o exemplo de um Sistema de Informação (SI), existe ainda uma documentação segmentada por níveis de acesso ao sistema, ou seja, a documentação é distribuída para os usuários de acordo com o papel que eles irão desempenhar operando o *software*.

Um dos tipos de documentação é a de caráter técnico, especifica para administradores de sistemas, onde normalmente aconselha-se o usuário a proceder com a leitura de arquivos contendo instruções de instalação e configurações iniciais, bem como as estruturas do BD e rotinas de manutenção.

Sendo assim, esperasse que haja determinado sigilo sobre a existência de alguns documentos, além de um rigoroso esquema de restrição ao acesso, permitindo apenas aos usuários com a devida autorização, tenham contato com conteúdo de caráter técnico estrutural, bem como os demais conteúdos de caráter gerencial.

* 1. GERENCIAMENTO DA DOCUMENTAÇÃO

Não menos importante que a produção dos documentos do *software*, é a tarefa de gerenciar o conteúdo, sendo este o objetivo final deste TG. Assim, gerir esse conteúdo consiste em juntar o que foi produzido, as vezes de modo descentralizado, e apresentar a documentação para o público interessado de forma simples, familiar, atraente, acessível, pesquisável, entre outras características.

Tudo isso deve ser feito considerando que pode haver a necessidade de editar a documentação a qualquer tempo e que as alterações devem ser disponibilizadas o mais breve possível, para que qualquer indivíduo que recorra a essa base de dados de especificações encontre informações atualizadas.

Sendo assim, a linguagem *Markdown* e as respectivas ferramentas para tratamento e produção de documentação que serão apresentadas no próximo capítulo, servirão como base para cumprir a proposta deste TG.

1. LINGUAGEM MARKDOWN

Graduado em Ciência da Computação, o norte americano John Gruber mantém na internet um *weblog*[[2]](#footnote-2) de tecnologia chamado *Daring Fireball* (BLANC, 2008). O foco principal deste *weblog,* ou simplesmente *blog*, como costuma-se dizer, é a escrita de artigos sobre: tecnologias empregadas no sistema operacional *MacOS,* da empresa *Apple;* aplicativos desenvolvidos para o sistema *MacOS*; e outros produtos da empresa *Apple*.

Além disso, ou talvez em função disso (escrever artigos para seu *blog*) John Gruber inventou o que se conhece como linguagem de marcação leve *Markdown*, que foi lançada em março de 2004 e visa simplificar o trabalho de escrita de textos para *web,* que são publicados no formato *HTML* (GRUBER, 2004).

Na página dedicada ao projeto, que faz parte do *blog Daring Fireball*, (GRUBER, 2017) define *Markdown* como “uma ferramenta de conversão de texto simples em texto no formato *HTML* para escritores *web*”.

Para melhor compreender aquilo que de fato vem a ser *Markdown*, épreciso contextualizar o problema que John Gruber enfrentava na época, que deu vasão a iniciativa de criação da linguagem.

Conforme o relato de (GRUBER, 2004), manipular as *tags*[[3]](#footnote-3) que compõe a sintaxe da linguagem *HTML* era uma tarefa fácil e, por esse motivo, ao escrever os artigos de seu *blog* elecostumava formatá-los diretamente com as *tags* *HTML*.

Apesar de ter facilidade para trabalhar com *HTML*, em determinado momento o criador da linguagem *Markdown* pôs-se a refletir sobre a morosidade do processo de escrita que antecedia suas postagens, que segundo ele consistia em: “(1) Escrever no *BBEdit*[[4]](#footnote-4); (2) Visualizar em um navegador; (3) Voltar para *BBEdit* e revisar o que foi escrito; (4) Repetir o processo até terminar; e (5) Fazer *login*[[5]](#footnote-5) no *MT*[[6]](#footnote-6), colar o artigo e publicar” (GRUBER, 2004).

Este processo contrastava com a opinião de (GRUBER, 2004) em relação ao uso de computadores. Segundo ele “a principal vantagem de usar um computador para escrever é o imediatismo da edição. Escreva, leia, revise, tudo na mesma janela, tudo no mesmo modo”.

Assim, Gruber percebeu que não havia vantagem em utilizar diretamente a formatação *HTML* para a criação dos artigos de seu *weblog,* chegando à seguinte conclusão:

Há uma razão pela qual navegadores de texto simples como o *Lynx* não mostram apenas o código-fonte *HTML* bruto. Simplesmente não é para ser um formato legível. Não lhe parece estranho escrever em um formato que não é legível? De repente, me pareceu absurdo (GRUBER, 2004).

A conclusão de John Gruber, em especial a referência ao navegador web *Lynx,* que é um navegador *web* em modo texto (LYNX, 2017), não parece adequada, considerando que o propósito para o qual existem navegadores *web* é para que o conteúdo marcado com *tags* *HTML* seja renderizado e apresentado como texto legível.

Certamente, na maioria dos casos, o objetivo almejado ao se fazer uso um navegador *web*, não se trata de observar o código fonte das páginas, ainda que esteja acessando as páginas através de navegadores *web* simples, como no caso do *Lynx.*

Na CLI do SO *Debian GNU/Linux,* exibida na Figura 16, vê-se a sintaxe do comando que faz uma chamada ao navegador web *Lynx* para exibição da página inicial do site *daringfireball.net.*

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 16** - Chamada ao navegador *Lynx*

**Fonte:** Autoria própria.

Já na Figura 17, pode-se observar o navegador *web* *Lynx* exibindo, em um terminal do SO *Debian GNU/Linux,* uma parte da página inicial do site *daringfireball.net.*

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 17** - Navegador *web* *Lynx* exibindo a página inicial do site *daringfireball.net*

**Fonte:** Autoria própria.

A partir do conteúdo exibido na Figura 17, constata-se que o navegador *Lynx* exibe a página inicial do site *daringfireball.net* renderizada de forma simples e não o código fonte (marcação *HTML*) da página, conforme John Gruber havia concluído. Ou seja, apesar de óbvio fica comprovado que a marcação *HTML* não tem como objetivo ser apresentada à humanos.

Em face a este contexto, (GRUBER, 2017) se sentiu motivado a criar a linguagem *Markdown,* que em uma segunda definição, um pouco mais elaborada que a primeira, pois dessa vez considera-se também a sintaxe da linguagem, ele afirma: “*Markdown* são duas coisas: (1) uma sintaxe de formatação de texto simples; e (2) uma ferramenta de software, escrita em, *Perl*[[7]](#footnote-7) que converte a formatação de texto simples para *HTML* e *XHTML*[[8]](#footnote-8) válido[[9]](#footnote-9)”.

Por fim, (GRUBER, 2017) concluí: “Um documento com formatado com *Markdown* deve ser publicado como é, como texto sem formatação, sem parecer que tenha sido marcado com *tags* ou instruções de formatação”.

* 1. LINGUAGEM DE MARCAÇÃO LEVE

Em sua dissertação de Mestrado (ALEXANDRE, 2017) cita que “o princípio das linguagens de marcação leve (em inglês *lightweight markup languages)* é que os textos sejam fáceis de serem digitados e lidos por humanos”. Assim, embora John Gruber não use exatamente essas palavras para definir seu invento, *Markdown* é considerada uma linguagem de marcação leve. O que é passível de constatação ao observar a sintaxe da linguagem que será demonstrada no tópico 3.3.

Em linhas gerais, o propósito de se utilizar *Markdown* é a conveniência de ter um texto estruturado de fácil compreensão para leitura, ao passo que a redação também seja facilitada pelo uso de uma sintaxe minimalista empregada na marcação de elementos tais como: parágrafos, títulos, citações, imagens, *links*, entre outros. Soma-se a isto, o fato de que no final do processo, através do uso de uma ferramenta de software para conversão, obtém-se o mesmo texto em formato *HTML*, pronto para a publicação.

Existem inúmeras linguagens de marcação leve disponíveis, duas delas que possuem certa relevância são a *reStructuredText,* muito utilizadapara documentação de projetos escritos em na linguagem *Python* e a *wikitexto*, utilizada principalmente para edições de artigos na *Wikipédia*.

* 1. ARQUIVOS *MARKDOWN*

Geralmente, os textos escritos com a sintaxe de formatação *Markdown* são salvos em arquivos com a extensão “.md”, porém, é possível usar a extensão “.markdown”. Arquivos com ambas as extensões podem ser consumidos por ferramentas de conversão para o formato *HTML*.

Em inglês, as ferramentas de software que executam a conversão do formato *Markdown* para o formato *HTML* são denominadas pela palavra *parser.*

* 1. SINTAXE

Para a definição da sintaxe de formatação da linguagem *Markdown*, (GRUBER, 2017) cita que contou com a ajuda de pares como: Aron Swartz, Nathaniel Irons, Dan Benjamin, Daniel Bogan e Jason Perkins. Além dessa ajuda, ele também relatou que obteve inspiração para elaboração da sintaxe no formato de *e-mail* de texto simples.

Tal formato consiste em um texto onde os pontos de destaque tais como títulos, seções do texto, palavras importantes e *links*, são sinalizados pelo uso de caracteres especiais ao seu redor, conforme pode-se observar na Figura 18.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 18** - Formato de *e*-*mail* de texto simples.

**Fonte:** (SMITH, 2014) *online*.

O referido formato de *e-mail* de texto simples, faz parte da estratégia de envio de *e-mail* *marketing,* na qual faz-se uso de uma funcionalidade presente em servidores de *e-mail*, o *Multi-part MIME (Multiporpouse Internet Mail Extension).*

Assim, envia-se as mensagens em dois formatos: *text/plain* e *text/html.* Deste modo, caso o cliente de *e-mail* do destinatário não seja capaz de exibir a mensagem no formato *text/html*, possivelmente contendo imagens e outros gráficos, exibirá a mensagem no formato *text/plain,* semelhante ao que foi apresentado na Figura 18.

A seguir, serão expostos exemplos de utilização da sintaxe de formatação da linguagem *Markdown.* Os exemplos irão abranger alguns dos elementos de um texto estruturado, expondo em quadros de duas colunas, respectivamente, o texto escrito em *Markdown* e os resultados em *HTML* obtidos após a conversão*.* Na sequência, será apresentada para cada exemplo uma figura contendo a visão do texto *HTML* renderizado pelo navegador.

O objetivo de tal demonstração será introduzir o leitor deste TG à um conjunto mínimo de elementos sintáticos utilizado com maior frequência em textos escritos com a linguagem *Markdown*.

Trata-se de um experimento para o qual foram utilizadas a especificação da linguagem *Ma*r*kdown* definida por (GRUBER, 2017) e a ferramenta de conversão *Dingus*, disponível *online* no *blog* *Daring Fireball*.

* + 1. Parágrafos e linhas

Em *Markdown* um parágrafo é constituído de uma ou várias linhas escritas consecutivamente podendo ou não conter caracteres de quebra de linha. Assim, para obter mais de um parágrafo no texto deve haver uma linha em branco, sem qualquer caractere imprimível, separando duas linhas com caracteres imprimíveis. Uma quebra de linha no texto HTML pode ser obtida separando as palavras por dois espaços.

O Quadro 4 exibe um exemplo de emprego dos parágrafos.

**Quadro 4** - *Markdown*: Sintaxe de formatação - Parágrafos e linhas.

|  |  |
| --- | --- |
| Texto em *Markdown* | Texto convertido para *HTML* |
| Este é o primeiro parágrafo.  Este é o segundo parágrafo.  E esta e a continuação do segundo parágrafo em outra linha.  Este é o terceiro parágrafo, desta vez com quebra de linha. | <p>Este é o primeiro parágrafo.</p>  <p>Este é o segundo parágrafo.  E esta e a continuação do segundo parágrafo em outra linha.</p>  <p>Este é o terceiro parágrafo<br>desta vez com quebra de linha.</p> |

**Fonte:** Autoria própria**.**

A Figura 19 apresenta três os parágrafos renderizados.

**Figura 19** - *Markdown*: Saída *HTML* renderizada - Parágrafos e linhas.

**Fonte:** Autoria própria.

* + 1. HTML incorporado

A especificação da sintaxe de formatação *Markdown* contempla um conjunto limitado de *tags HTML,* por esse motivo, quando se pretende usar um elemento textual presente na sintaxe *HTML*, que não foi implementado em *Markdown*, pode-se inserir diretamente as *tags* *HTML*.

Para utilização de elementos de blocos tais como as *tags*: *<div>, <table>, <ul> e <pre>,* deve se respeitar a separação do bloco formatado com *tags HTML* do restante do conteúdo formatado com *Markdown,* utilizando uma linha em branco conforme demonstrado no Quadro 5.

**Quadro 5** - *Markdown*: Sintaxe de formatação - *HTML* em bloco.

|  |  |
| --- | --- |
| Texto em *Markdown* | Texto convertido para *HTML* |
| Este é um parágrafo.  <table border="1">  <tr>  <td>Markdown</td>  <td>HTML</td>  </tr>  <tr>  <td>Formato para escrita</td>  <td>Formato para publicação</td>  </tr>  </table>  Este é outro parágrafo. | <p>Este é um parágrafo.</p>  <table border="1">  <tr>  <td>Markdown</td>  <td>HTML</td>  </tr>  <tr>  <td>Formato para escrita</td>  <td>Formato para publicação</td>  </tr>  </table>  <p>Este é outro parágrafo.</p> |

**Fonte:** Autoria própria.

O resultado renderizado, onde são apresentados um parágrafo, uma tabela e outro parágrafo, pode ser observado na Figura 20.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 20** - *Markdown:* Saída *HTML* renderizada - *HTML* emBloco.

**Fonte:** Autoria própria.

Para utilização de elementos em linha tais como as *tags*: *<span>, <small>, <cite>* e *<del>* não existe restrição de posicionamento, podendo ser inseridos juntamente com linhas formatadas com sintaxe *Markdown*, conforme demonstrado no Quadro 6.

**Quadro 6** - *Markdown*: Sintaxe de formatação - *HTML* em linha.

|  |  |
| --- | --- |
| Texto em *Markdown* | Texto convertido para *HTML* |
| Este é um parágrafo com <small>marcação HTML em linha</small>.  Este é outro parágrafo escrito por <cite>Getúlio Vinicius</cite>. | <p>Este é um parágrafo com <small>marcação HTML em linha</small>.</p>  <p>Este é outro parágrafo escrito por<cite>Getúlio Vinicius</cite>.</p> |

**Fonte:** Autoria própria.

O resultado renderizado, onde são apresentados dois parágrafos contendo elementos *HTML* em linha pode ser observado na Figura 21.

**Figura 21** - *Markdown*: Saída *HTML* renderizada - *HTML* em linha.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** Autoria própria.

Uma observação importante em relação a sintaxe *Markdown* original e o *parser* escrito em *Perl* que foi disponibilizado por John Gruber, é que estes permitem a utilização de *tags* *HTML* mesmo para aqueles elementos que foram especificados. Assim, por exemplo, caso se opte por utilizar a *tag* *<img>* da sintaxe de formatação *HTML,* ao invés de usar a sintaxe definida para a linguagem *Markdown* em relação a inserção de imagens, que será vista no tópico 3.3.8, o *parser* irá processar normalmente a conversão.

* + 1. Ênfase

A sintaxe *Markdown* permite o emprego de ênfase de dois modos distintos, correspondendo respectivamente as *tags <em>* e *<strong>* da sintaxe *HTML.* Os caracteres utilizados para obter a formatação são “\_” (*underscore*) e “\*” (asterisco).

Para obter o efeito de ênfase indicando um contraste implícito ou explicito, o texto a ser destacado deve ser envolvido com 1 (um) caractere em cada extremidade, qualquer que seja entre os dois aceitos. E, do mesmo modo, para obter o efeito de ênfase indicando que a palavra ou o trecho possui grande importância, o texto a ser destacado deve ser envolvido com 2 (dois) caracteres em cada extremidade.

O Quadro 7 demonstra o emprego dos dois tipos de ênfase com a linguagem *Markdown.*

**Quadro 7** - *Markdown*: Sintaxe de formatação - Ênfase.

|  |  |
| --- | --- |
| Texto em *Markdown* | Texto convertido para *HTML* |
| No meu tempo existiam os profissionais chamados \_Webmaster\_. Hoje em dia são chamados de desenvolvedores \*front-end\*.  \_\_Documentar\_\_ o projeto de software é fundamental para o \*\*correto desenvolvimento\*\*. | <p>No meu tempo existiam os profissionais chamados <em>Webmaster</em>. Hoje em dia são chamados de desenvolvedores <em>front-end</em>.</p>  <p><strong>Documentar</strong> o projeto de software é fundamental para o <strong>correto desenvolvimento</strong>. </p> |

**Fonte:** Autoria própria.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaO resultado renderizado, onde é possível observar o emprego dos dois tipos de ênfase é apresentado na Figura 22.

**Figura 22** - *Markdown*: Saída *HTML* renderizada - Ênfase.

**Fonte:** Autoria própria.

O emprego de ênfase na estruturação de texto para a *web* geralmente provoca confusão em profissionais, sejam eles programadores – as vezes iniciantes, ou ainda escritores jornalistas, blogueiros (pessoas que escrevem artigos para *blogs*) entre outros.

A confusão se dá principalmente pelo resultado obtido quando o texto é renderizado. A ênfase empregada pelo uso da *tag HTML <em>,* geralmente é apresentada em formato de fonte itálico, enquanto que a ênfase empregada pelo uso da *tag HTML* *<strong>,* geralmente é apresentada em formato de fonte negrito.

A obtenção de negrito e itálico com *tags HTML* se dá respectivamente pelo uso de outras duas *tags* distintas: *<i>* e *<b>.* Estas não carregam consigo nenhum tipo de apelo semântico em relação ao texto como a indicação de contraste e grau de importância, segundo a documentação da linguagem *HTML* oferecida pela (MDN WEB DOC - MOZILLA, 2017) e (MDN WEB DOC - MOZILLA, 2017)

* + 1. Cabeçalhos

Para obtenção de cabeçalhos com a linguagem *Markdown*, pode-se recorrer a dois modelos distintos: *setext*[[10]](#footnote-10) e *atx*[[11]](#footnote-11).

O primeiro modelo, *setext*, permite apenas cabeçalhos de nível 1 e nível 2, e consiste em redigir o texto do cabeçalho sem qualquer marcação especial, em uma linha, e, na linha seguinte sublinhar o texto com os caracteres: “=” (igual) para cabeçalho de nível 1; e “-“ (hífen) para cabeçalho de nível 2. É necessário ao menos um caractere sublinhando o texto do cabeçalho e a linha do sublinhado pode conter apenas caracteres referente ao nível de cabeçalho desejado, conforme pode ser visto no Quadro 8.

**Quadro 8** - *Markdown*: Sintaxe de formatação - Cabeçalhos do modelo *setext*.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Este é um cabeçalho de nível 1 – h1  ============================  Este é um cabeçalho de nível 2 – h2  ------------------------------------------------- | <h1>Este é um cabeçalho de nível 1 – h1</h1>  <h2>Este é um cabeçalho de nível 2 – h2</h2> |

**Fonte:** Autoria própria.

O resultado renderizado, onde são apresentados dois cabeçalhos, pode ser visto na Figura 23.

**Figura 23** - *Markdown*: Saída *HTML* renderizada - Cabeçalho modelo *setext*.

**Fonte:** Autoria própria.

O segundo modelo, *atx*, permite a utilização dos seis níveis de cabeçalho implementados nas especificações do *HTML*. Consiste e marcar o início da linha que contém o texto do cabeçalho com a quantidade de caracteres “#” (tralha) referente ao nível de cabeçalho que se deseja obter, sendo 1 (um) caractere para cabeçalho de nível 1, 2 (dois) caracteres para cabeçalho de nível 2 e assim sucessivamente até o cabeçalho de nível 6, conforme pode-se observar no Quadro 9.

**Quadro 9** - *Markdown*: Sintaxe de formatação - Cabeçalho no modelo *atx*.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| # Este é um cabeçalho de nível 1 – h1  ## Este é um cabeçalho de nível 2 – h2  ### Este é um cabeçalho de nível 3 – h3  #### Este é um cabeçalho de nível 4 – h4  ##### Este é um cabeçalho de nível 5 – h5  ###### Este é um cabeçalho de nível 6 – h6 | <h1>Este é um cabeçalho de nível 1 – h1</h1>  <h2>Este é um cabeçalho de nível 2 – h2</h2>  <h3>Este é um cabeçalho de nível 3 – h3</h3>  <h4>Este é um cabeçalho de nível 4 – h4</h4>  <h5>Este é um cabeçalho de nível 5 – h5</h5>  <h6>Este é um cabeçalho de nível 6 – h6</h6> |

**Fonte**: Autoria própria.

O resultado renderizado, onde são apresentados os seis níveis de cabeçalho, pode ser visto na Figura 24.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 24** - *Markdown*: Saída *HTML* renderizada - Cabeçalho modelo atx.

**Fonte:** Autoria própria.

* + 1. Listas

Obter listas em um texto formatado com a sintaxe *Markdown* é extremamente simples. Blocos de listas, quando precedidos por texto corrido, devem ser iniciados após uma linha em branco, delimitando a separação da lista do parágrafo que a antecede. O mesmo critério vale para parágrafos que sucedem uma lista.

Para listas ordenadas basta utilizar os caracteres numéricos sucedidos por um ponto no início da linha, conforme demonstrado no Quadro10.

**Quadro 10** - *Markdown*: Sintaxe de formatação - Listas ordenadas.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| 1. Primeiro item da lista.  2. Segundo item da lista.  3. Terceiro item da lista.  4. Quarto...  Este parágrafo sucede uma lista ordenada e por isso existe uma linha em branco separando-o da lista.  Este parágrafo antecede uma lista ordenada e por isso existe uma linha em branco abaixo separando-o da lista.  1. Primeiro item da lista.  2. Segundo item da lista. | <ol>  <li>Primeiro item da lista.</li>  <li>Segundo item da lista.</li>  <li>Terceiro item da lista.</li>  <li>Quarto...</li>  </ol>  <p>Este parágrafo sucede uma lista ordenada e por isso existe uma linha em branco separando-o da lista.</p>  <p>Este parágrafo antecede uma lista ordenada e por isso existe uma linha em branco abaixo separando-o da lista.</p>  <ol>  <li>Primeiro item da lista.</li>  <li>Segundo item da lista.</li>  </ol> |

**Fonte:** Autoria própria.

O resultado renderizado pode ser observado na Figura 25, onde são apresentadas duas listas e dois parágrafos.

Uma imagem contendo captura de tela

Descrição gerada com alta confiança**Figura 25** - *Markdown*: Saída *HTML* renderizada - Listas ordenadas.

**Fonte:** Autoria própria.

Para listas não ordenadas os critérios de separação de blocos e parágrafos seguem inalterados, mas os caracteres de início da linha que constitui um item da lista são diferentes. Pode-se utilizar, inclusive de forma mesclada, os caracteres “+” (sinal de adição), “-“ (hífen) e “\*” (asterisco), conforme demonstrado no Quadro 11.

**Quadro 11** - *Markdown*: Sintaxe de formatação - Listas não ordenadas.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Exemplo com o caractere “+”:  + Primeiro item da lista.  + Segundo item da lista.  Exemplo com o caractere “-”:  - Primeiro item da lista.  - Segundo item da lista.  Exemplo com o caractere “\*”:  \* Primeiro item da lista.  \* Segundo item da lista.  Exemplo com os caracteres mesclados:  + Primeiro item da lista.  - Segundo item da lista.  \* Terceiro item da lista. | <p>Exemplo com o caractere “+”:</p>  <ul>  <li>Primeiro item da lista.</li>  <li>Segundo item da lista.</li>  </ul>  <p>Exemplo com o caractere “-”:</p>  <ul>  <li>Primeiro item da lista.</li>  <li>Segundo item da lista.</li>  </ul>  <p>Exemplo com o caractere “\*”:</p>  <ul>  <li>Primeiro item da lista.</li>  <li>Segundo item da lista.</li>  </ul>  <p>Exemplo com os caracteres mesclados:</p>  <ul>  <li>Primeiro item da lista.</li>  <li>Segundo item da lista.</li>  <li>Terceiro item da lista.</li>  </ul> |

**Fonte**: Autoria própria.

A Figura 26 mostra o resultado renderizado do primeiro exemplo do Quadro 11, uma lista não ordenada com dois itens construída com o caractere “+”.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 26** - *Markdown*: Saída *HTML* renderizada - Lista não ordenada.

**Fonte:** Autoria própria.

É possível obter listas encadeadas, níveis de lista, tanto para listas ordenadas quanto para listas não ordenadas ou, de forma conjunta mesclando os dois tipos de listas.

Assim, para obter uma lista de nível 2, imediatamente após um item da lista de nível 1, deve-se aplicar um recuo com o caractere não imprimível de tabulação, utilizando a tecla “tab” e então as próximas linhas que constituírem um item da lista de nível 2 também deverão ser precedidas do mesmo caractere de tabulação.

Um elemento de uma lista de nível 3 deve ser precedido por dois caracteres de tabulação, e assim sucessivamente para os demais níveis, conforme demonstrado no Quadro 12.

**Quadro 12** - *Markdown*: Sintaxe de formatação - Listas ordenadas e não ordenadas.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Lista ordenada:  1. Primeiro item da lista de nível 1.  2. Segundo item da lista de nível 1.  1. Primeiro item da lista de nível 2.  2. Segundo item da lista de nível 2.  Lista não ordenada:  \* Primeiro item da lista de nível 1.  \* Primeiro item da lista de nível 2.  \* Segundo item da lista de nível 2.  \* Segundo item da lista de nível 1.  Listas ordenadas e não ordenadas:  1. Primeiro item da lista de nível 1.  2. Segundo item da lista de nível 1.  \* Primeiro item da lista de nível 2.  1. Primeiro item da lista de nível 3.  2. Segundo Item da lista de nível 3.  \* Segundo item da lista de nível 2.  3. Item da lista de nível 1. | <p>Lista ordenada:</p>  <ol>  <li>Primeiro item da lista de nível 1.</li>  <li>Segundo item da lista de nível 1.  <ol><li>Primeiro item da lista de nível 2.</li>  <li>Segundo item da lista de nível 2.</li></ol></li>  </ol>  <p>Lista não ordenada:</p>  <ul>  <li>Primeiro item da lista de nível 1.  <ul><li>Primeiro item da lista de nível 2.</li>  <li>Segundo item da lista de nível 2.</li></ul></li>  <li>Segundo item da lista de nível 1.</li>  </ul>  <p>Listas ordenadas e não ordenadas:</p>  <ol>  <li>Primeiro item da lista de nível 1.</li>  <li>Segundo item da lista de nível 1.  <ul><li>Primeiro item da lista de nível 2.  <ol><li>Primeiro item da lista de nível 3.</li>  <li>Segundo Item da lista de nível 3.</li></ol></li>  <li>Segundo item da lista de nível 2.</li></ul></li>  <li>Item da lista de nível 1.</li>  </ol> |

**Fonte:** Autoria própria.

A Figura 27 exibe o resultado daquilo que foi demonstrado no Quadro 12. Uma lista ordenada com dois níveis, uma lista não ordenada com dois níveis e uma lista que mescla listas ordenadas e não ordenadas com três níveis.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura** **27** - *Markdown*: Saída *HTML* renderizada - Listas oredenadas e não ordenadas.

**Fonte:** Autoria própria.

* + 1. Citações em bloco

A existência de um elemento para formatação de bloco de citação na sintaxe da linguagem *Markdown*, denota claramente a intenção de John Gruber em criar uma ferramenta para simplificar o trabalho daqueles que escrevem texto para a internet, sendo que citações são elementos que se observa mais frequentemente em artigos de *blogs* e sites de notícias.

A tarefa de destacar um texto como citação em bloco usando *Markdown* consiste em iniciar a linha com um caractere “>” (maior que) e, para interromper a sequência do bloco de citação basta inserir um parágrafo normal, ou seja, uma linha em branco conforme a demonstração constante no Quadro 13.

**Quadro 13** - *Markdown*: Sintaxe de formatação - Citação em Bloco.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Este texto constitui um parágrafo normal.  Abaixo será mostrado um exemplo de citação em bloco:  > Este texto é um exemplo de como funciona a citação em bloco.  Um bloco de citação pode conter várias linhas.  Este é um outro parágrafo. Abaixo virá uma citação com dois parágrafos:  > Este é o primeiro parágrafo da citação.  > Citação em bloco pode conter vários parágrafos. | <p>Este texto constitui um parágrafo normal.  Abaixo será mostrado um exemplo de citação em bloco:</p>  <blockquote>  <p>Este texto é um exemplo de como funciona a citação em bloco.  Um bloco de citação pode conter várias linhas.</p>  </blockquote>  <p>Este é um outro parágrafo. Abaixo virá uma citação com dois parágrafos:</p>  <blockquote>  <p>Este é o primeiro parágrafo da citação.</p>  <p>Citação em bloco pode conter vários parágrafos.</p>  </blockquote> |

**Fonte:** Autoria própria.

O resultado renderizado, onde são apresentados dois blocos de citação, pode ser visto na Figura 28.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 28** - *Markdown*: Saída *HTML* renderizada - Citação em bloco.

**Fonte:** Autoria Própria.

Outra possibilidade de uso contemplada pela sintaxe de formatação da linguagem *Markdown*, para blocos de citação, é a composição da estrutura em conjunto com outros elementos já apresentados, tais como: listas, cabeçalhos e destaques de texto.

O Quadro 14 traz uma demonstração de bloco de citação composto por mais elementos da sintaxe *Markdown*.

**Quadro 14** - *Markdown*: Sintaxe de formatação - Citação em Bloco composta por mais elementos.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Abaixo será mostrado um exemplo de citação em bloco com vários elementos:  > # Este é o título da citação  > Este texto é o primeiro parágrafo da citação.  > 1. Aqui um \*\*elemento\*\* de lista ordenada.  > 2. Aqui outro \*\*elemento\*\* de lista ordenada.  > + Aqui um \*\*elemento\*\* de lista não ordenada.  > + Aqui outro \*\*elemento\*\* de lista não ordenada.  > \_Este é o último parágrafo da citação.\_ | <p>Abaixo será mostrado um exemplo de citação em bloco com vários elementos:</p>  <blockquote>  <h1>Este é o título da citação</h1>  <p>Este texto é o primeiro parágrafo da citação.</p>  <ol>  <li>Aqui um <strong>elemento</strong> de lista ordenada.</li>  <li>Aqui outro <strong>elemento</strong> de lista ordenada.  <ul><li>Aqui um <strong>elemento</strong> de lista não ordenada.</li>  <li>Aqui outro <strong>elemento</strong> de lista não ordenada.</li></ul></li>  </ol>  <p><em>Este é o último parágrafo da citação.</em></p>  </blockquote> |

**Fonte:** Autoria própria.

A Figura 29 exibe o resultado renderizado do código demonstrado no Quadro 14.

**Figura 29** - *Markdown*: Saída *HTML* renderizada - Citação em bloco composta por mais elementos.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** Autoria própria.

* + 1. Links

Quanto aos *links*, a linguagem *Markdown* implementa o recurso utilizando duas sintaxes de formatação distintas: *inline* e por referência.

A sintaxe *inline* consiste em envolver entre colchetes “[]” o texto chamado de ancora, que após a renderização do código será exibido na página ao invés de exibir o endereço para onde o aponta o *link* e imediatamente após o endereço (caminho para o arquivo) envolvido por parênteses “()”.

O endereço corresponde ao atributo *href* da *tag HTML <a>*, e,opcionalmente pode-se informar um título para o *link* que corresponderá ao atributo *title* da *tag <a>*. O título deve ser posto entre aspas, separado do endereço por um caractere de espaço e envolvido pelo mesmo conjunto de parênteses do endereço.

O endereço pode ser local ou remoto (da rede ou *internet*). Sendo um arquivo remoto, deve-se indicar a *URL - Uniform Resource Location* (em português, Localizador Uniforme de Recursos), ou seja, o endereço de rede pelo qual se acessa um recurso, neste caso o arquivo ou página que se deseja referenciar através do link. Sendo um endereço local, para um arquivo armazenado no mesmo servidor, pode-se usar um caminho relativo a partir do diretório corrente onde será armazenado o arquivo de extensão “.md” que contém o *link*.

O Quadro 15 traz uma demonstração acerca do uso de *links* *inline* com a linguagem Markdown.

**Quadro 15** - *Markdown*: Sintaxe de formatação – *Links inline*.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Aqui temos um [exemplo] (http://examplo.com/ "Link de exemplo") de link inserido em um texto.  [Este link](http://example.net/) não possui um atributo \_title\_.  Este [outro link](/outra-pagina/) aponta para um recurso armazenado no mesmo servidor. | <p>Aqui temos um <a href="http://examplo.com/" title="Link de exemplo">exemplo</a> de link inserido em um texto.</p>  <p><a href="http://example.net/">Este link</a> não possui um atributo <em>title</em>.</p>  <p>Este <a href="/outra-pagina/">outro link</a> aponta para um recurso armazenado no mesmo servidor.</p> |

**Fonte:** Autoria própria.

A Figura 30 exibe o exemplo renderizado, do demonstrado no Quadro 15 renderizado.

**Figura 30** - *Markdown*: Saída *HTML* renderizada – *Links inline*.

**Fonte:** Autoria própria.

A sintaxe de *links* por referência pode ser usada para simplificar a leitura do texto escrito em *Markdown*, isto porque diminui a quantidade de elementos de sintáticos e endereços longos no corpo do texto, substituindo-os por chaves curtas que os identificam em um bloco de texto separado na redação.

Para inserir *links* através de referências, deve-se envolver o texto de ancora em colchetes “[]” e logo na sequência, envolvido por um novo par de colchetes deve-se informar a chave de referência do *link*.

Em um bloco separado do parágrafo, ou qualquer outro elemento que contenha o *link*, deve-se construir a referência, que consiste em: uma linha iniciada pela chave de identificação envolta por colchetes; sucedida pelo caractere “:” (dois pontos); um caractere de espaço; o endereço (local ou remoto) referente ao atributo *href* da *tag HTML <a>*; e por fim, opcionalmente, o título do link referente ao atributo *title* da *tag <a>* entre aspas.

O Quadro 16 demonstra o uso de links por referência na linguagem *Markdown.*

**Quadro 16** - *Markdown*: Sintaxe de formatação - *Links* por referência.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Aqui temos um [exemplo][ex1] de link inserido por referência.  [Este link][ex2], também inserido por referência, não possui um atributo \_title\_.  Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado.  [ex1]: http://exemplo.com "Título de exemplo"  [ex2]: /pagina-de-exemplo/ | <p>Aqui temos um <a href="http://exemplo.com" title="Título de exemplo">exemplo</a> de link inserido por referência.</p>  <p><a href="/pagina-de-exemplo/">Este link</a>, também inserido por referência, não possui um atributo <em>title</em>.</p>  <p>Aqui temos apenas um parágrafo. A indicação dos endereços vem logo abaixo e não aparece no texto renderizado.</p> |

**Fonte:** Autoria própria.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaA Figura 31 exibe o resultado renderizado da marcação *Markdown* elaborada no Quadro 16.

**Figura 31** - *Markdown*: Saída *HTML* renderizada – *Links* por referência.

**Fonte:** Autoria própria.

* + 1. Imagens

O recurso de inserção de imagens através da linguagem *Markdown* possui sintaxe semelhante a inserção de *links*. A diferenças aqui consiste na existência de um ponto de exclamação “!” precedendo todo o restante da marcação, que é idêntica a formatação de links.

No caso de inserção de imagens, o texto envolto por colchetes, que na inserção de links representa o texto ancora, desta vez representa o atributo *alt* da *tag HTML <img>*. Na sequência, o texto entre parênteses corresponde respectivamente a *url* do arquivo de imagem, representando o atributo *src* da *tag* *<img>* e, após a separação por caractere de espaço, o título da imagem entre aspas, que representa o atributo *title* da *tag <img>.*

O Quadro 17 exibe a formatação necessária para inserção de imagens a partir sintaxe da linguagem *Markdown.*

**Quadro 17** - *Markdown*: Sintaxe de formatação - Imagens.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Inserção de uma imagem externa ao site a partir de uma url:  ![Daring Fireball - Logo](https://daringfireball.net/graphics/logos/ "Logo do site Daring Fireball")  Inserção de uma imagem local a partir do caminho relativo:  ![Imagem - Local 1](/imagens/logo1 "Título 1")  Inserção de uma imagem por referência:  Imagem - Local 2][2]  [2]: /imagens/logo2 "Título 2" | <p>Inserção de uma imagem externa ao site a partir de uma url:</p>  <p><img src="https://daringfireball.net/graphics/logos/" alt="Daring Fireball - Logo" title="Logo do site Daring Fireball" /></p>  <p>Inserção de uma imagem local a partir do caminho relativo:</p>  <p><img src="/imagens/logo1" alt="Imagem - Local 1" title="Título 1" /></p>  <p>Inserção de uma imagem por referência:</p>  <p><img src="/imagens/logo2" alt="Imagem - Local 2" title="Título 2" /></p> |

**Fonte:** Autoria própria.

A Figura32 exibe o resultado renderizado a partir da formatação demonstrada no Quadro 17.

**Figura 32** - *Markdown*: Saída *HTML* renderizada - Imagens.

**![Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** Autoria própria.

É possível observar na Figura 32 que ao invés das imagens *logo1* e *logo2*, a rederização apresentou o texto alternativo, que foi definido entre colchetes na formatação. Isto porque os respectivos arquivos não foram localizados no servidor onde a ferramenta utilizada para promover a conversão do formato *Markdown* para o formato *HTML*, *Dingus,* está instalada.

* + 1. Código

*Markdown* possui sintaxe de formatação para exibição de código, que representa a *tag HTML <code>* e pode ser inserida *inline*, ou seja, destacando um trecho do parágrafo como fragmento de código, ou ainda, como elemento de bloco de texto pré-formatado, onde o conteúdo da *tag <code>* é encapsulado pela *tag <pre>* da marcação *HTML*.

Segundo a documentação de marcação *HTML* oferecida pela (MDN WEB DOC - MOZILLA, 2017) “a *tag HTML* *<code>* é utilizada para representar um fragmento de código de computador. Por padrão, o conteúdo da *tag* *<code>* é exibido pelos navegadores de internetusando fonte mono espaçada”.

Para obtenção de um fragmento de código em um parágrafo qualquer é preciso envolver o trecho com o caractere “`” (apostrofe). E para a obtenção de um bloco de código, deve-se separar o trecho do texto normal com uma linha em branco e então cada linha pertencente ao código deve ser precedida por um caractere de tabulação.

O Quadro 18 demonstra a sintaxe de formatação *Markdown* para exibição de código.

**Quadro 18** - *Markdown*: Sintaxe de formatação - Código.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| Para exibir um bloco de código em HTML pode-se utilizar as tags `<pre>` e `<code>`.  Abaixo um trecho de código em bloco:  <h1>Exemplo de exibição de código</h1>  <div>  <p>Um parágrafo</p>  </div>    <a href="www.um-site-de-exemplo.com">Um link de exemplo</a> | <p>Para exibir um bloco de código em HTML pode-se utilizar as tags <code>&lt;pre&gt;</code> e <code>&lt;code&gt;</code>.</p>  <p>Abaixo um trecho de código em bloco:</p>  <pre><code>&lt;h1&gt;Exemplo de exibição de código&lt;/h1&gt;  &lt;div&gt;  &lt;p&gt;Um parágrafo&lt;/p&gt;  &lt;/div&gt;  &lt;a href="www.um-site-de-exemplo.com"&gt;Um link de exemplo&lt;/a&gt;  </code></pre> |

**Fonte:** Autoria própria.

O resultado renderizado, onde pode ser observada a exibição de código *inline* (em linha) e de um bloco e código é apresentado na Figura 33.

**Figura 33** - *Markdown*: Saída *HTML* renderizada - Código.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** Autoria própria.

* + 1. Caractere de escape

A respeito da sintaxe *Markdown,* o último exemplo em relação a definição de John Gruber será o caractere de escape. Em algumas situações o que se deseja é a representação fidedigna de um caractere e não que o mesmo seja interpretado pelo *parser Markdown* como sendo parte de uma marcação sintática. Nestes casos é preciso informar o *parser* que o caractere seguinte faz parte do contexto da redação. Para que essa informação seja considerada pelo *parser* é necessário que o caractere que se deseja representar seja precedido por um caractere de escape. Para a linguagem *Markdown* John Gruber definiu que este caractere seja a “\” (contra barra).

O Quadro 19 demonstra a inserção de caracteres utilizando caractere de escape, bem como uma listagem de todos os caracteres que para serem representados literalmente devem ser escapados.

**Quadro 19** - *Markdown*: Sintaxe de formatação - Caractere de escape.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| + \\ "Contra barra  + \` "Apóstrofe"  + \\* "Asterisco"  + \\_ "Sublinhado"  + \{\} "Chaves"  + \[\] "Colchetes"  + \(\) "Parênteses"  + \# "Tralha"  + \+ "Sinal de adição"  + \- "Hífen"  + \. "ponto"  + \! "Ponto de exclamação" | <ul>  <li>\ "Contra barra</li>  <li>` "Apóstrofe"</li>  <li>\* "Asterisco"</li>  <li>\_ "Sublinhado"</li>  <li>{} "Chaves"</li>  <li>[] "Colchetes"</li>  <li>() "Parênteses"</li>  <li># "Tralha"</li>  <li>+ "Sinal de adição"</li>  <li>- "Hífen"</li>  <li>. "ponto"</li>  <li>! "Ponto de exclamação"</li>  </ul> |

* 1. *PARSERS*
     1. *Paser* criado por John Gruber

A implementação em feita por John Gruber, escrita com a linguagem de programação *Perl,* foi o primeiro *parser* para *Markdown* (MACFARLANE, GREENSPAN, *et al.*, 2017) e pode ser utilizado em conjunto com aplicativos para publicação de conteúdo web tais como: *Movable Type* e *BBEdit.* citados por John Gruber.

Tal software está disponível para *Download* na página do projeto e licenciado como software livre sob os termos de uma licença no estilo *BSD open source*[[12]](#footnote-12). Sua versão atual é a 1.0.1, sendo que a última atualização no código ocorreu em 17 de dezembro de 2004 (GRUBER, 2017).

Este *parser* também pode ser utilizado na CLI de sistemas operacionais *GNU/Linux* para converter textos, como no exemplo da Figura 34, onde através de uma chamada ao interpretador da linguagem *Perl* executa-se o *script Markdown.pl* passando como parâmetro de conversão *–html4tags* que indica a versão da linguagem *HTML* que será usada na conversão (naquela época era a versão 4), e, por fim, o nome do arquivo que contém o texto a ser convertido, no caso *code.md.*

Antes da conversão, é exibido o conteúdo do arquivo *code.md* utilizado o comando *cat* do SO *Debian GNU/Linux.*

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança**Figura 34 - Conversão de texto em *Markdown* para *HTML* utilizando *Markdown*.*pl* 1.0.1.

**Fonte:** Autoria própria.

* + 1. Novos *parsers*

A sintaxe de formatação *Markdown* foi implementada em várias linguagens de programação, dando origem a outros *softwares* de conversão. Tais softwares, além de converterem textos escritos em *Markdown* para o formato *HTML* e até mesmo outros formatosde maneira solo, por meio de suas próprias interfaces de entrada e saída, também podem ser utilizados em conjunto com outras aplicações, ou seja, servindo como *plugin* (complemento que estende as funcionalidades a um aplicativo).

Um bom exemplo é o *software Pandoc*, desenvolvido com a linguagem de programação *Haskell*, que permite realizar a conversão de texto escrito em diversos formatos de arquivo em outros diversos formatos arquivo. Este software possui uma implementação para conversão de textos *Markdown*, não apenas para HTML, mas também para outros formatos, tais como: *PDF, DOCX e TXT* (MACFARLANE, 2017).

A Figura 35 demonstra o procedimento para conversão, utilizando o *Pandoc*, de um arquivo em formato *Markdown* para o formato *DOCX* (formato de arquivo padrão do editor de textos *Microsoft Word*).

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 35** - Conversão de texto no formato *Markdown* para *DOCX* com o software *Pandoc*.

**Fonte:** Autoria própria.

Na demonstração, feita através do terminal do SO *Debian GNU/Linux*, inicialmente é listado o conteúdo do diretório corrente inserido na CLI o comando *ls*; na sequência o conteúdo do arquivo a ser convertido (teste-1.md) é impresso na tela com o uso do comando *cat*; então o comando para conversão, *pandoc*, é executado recebendo como parâmetros: o nome do arquivo original, a opção *-s* que indica a criação de um único arquivo e a opção *-o* que especifica o nome do arquivo de saída inserido no fim (teste-1.docx); por fim o conteúdo do diretório corrente é listado novamente, a fim de verificar a criação do arquivo com o texto convertido.

A Figura 36 exibe o conteúdo do arquivo convertido (teste-1.docx) no editor de textos *LibreOffice Writer*, que possibilita a leitura de arquivos de texto no formato DOCX.

**Figura 36** - Texto convertido pelo Pandoc de Markdown para DOCX.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** Autoria própria.

Existe uma vasta lista de *parses* e extensões para *Markdown*, sendo que, algumas das implementações notáveis são:

* ***PHP Markdown* e *PHP Markdown Extra*:** Uma biblioteca e uma extensão para a biblioteca criada por Michel Fortin utilizando a linguagem de programação *PHP,* que possibilitam a utilização de *Markdown* para postagem de conteúdo em sistemas *web* com a conversão sendo realizada em tempo de execução das aplicações (FORTIN, 2018). A biblioteca *PHP Markdown Extra* amplia a quantidade de recursos sintáticos da especificação original da linguagem *Markdown*.
* **Python Markdown:** Uma biblioteca criada utilizando a linguagem de programação *Python*, que além da implementação original da sintaxe *Markdown,* possibilita, através de uma API – (em português,), a integração de diversas extensões que ampliam os recursos sintáticos especificados por John Gruber, (THE PYTHON-MARKDOWN PROJECT, 2017). Os primeiros autores do projeto são: Waylan Limberg, Yuri Takteyev, Manfed Stienstra, Artem Yunusov e David Wolever.
* **Kramdown:** Uma biblioteca escrita por Thomas Leitner com a linguagem de programação *Ruby*. Ela implementa a sintaxe original de *Markdown* e possibilita o uso de extensões que ampliam a quantidade de recursos (LEITNER, 2017).
  + 1. Recursos implementados pelos novos *parsers*

Retomando a questão da sintaxe, após o lançamento da ferramenta criada por John Gruber, os *parsers* que foram lançados em diversas linguagens de programação agregaram novas funcionalidades na linguagem *Markdown*, tais como: tabelas, notas de rodapé, listas de definição, diagramas UML e outros.

Não cabe neste TG, demonstrar cada uma dessas novidades, uma vez que já foi realizada uma introdução a sintaxe básica que abrange a maior parte das funcionalidades da linguagem *Markdown*.

Entretanto, um recurso muito utilizado em textos estruturados são as tabelas, que possibilitam a apresentação de dados de maneira simples, favorecendo o aspecto analítico de um documento. Portanto, será feita uma exceção para este recurso que será demonstrado utilizando o *parser* *PHP Markdown Extra* como exemplo, sendo que, a conversão para *HTML* e renderização da marcação *HTML,* foram realizadas utilizando a aplicação *online* *Babelmark 2*.

Assim, para que se obtenha uma tabela formatada com *Markdown*, é necessário envolver o conteúdo, ou seja, os dados que serão apresentados nas células da tabela por caracteres “|” (pipe). Além disso, sabe-se que em *HTML* é possível indicar as células que compõe o cabeçalho da tabela, que corresponde ao título de cada coluna, e as células que compõe o corpo da tabela, onde cada linha corresponde analogamente a um registro exposto na tabela.

Para reproduzir este efeito com a sintaxe *Markdown*, é preciso inserir uma linha especial na estrutura, especificamente a segunda linha da marcação, de modo que a primeira linha corresponderá ao título das colunas, e a partir da terceira linha serão inseridos os dados.

Na segunda linha, o conteúdo disposto entre os caracteres “|” (pipe), deve ser composto por: um caractere “ “ (espaço); na sequência, uma cadeia de caracteres “-“ (hífen), não existindo limite determinado de caracteres que podem ser inseridos; e por fim, após essa cadeia e antes do novo caractere “|” (pipe) é preciso inserir um caractere “ “ (espaço). Deste modo o texto de cada coluna seguirá o alinhamento padrão das tabelas, ou seja, alinhado à esquerda.

É possível determinar o alinhamento que o texto de cada coluna seguirá inserindo um caractere “:” (dois pontos) em lugar dos caracteres “ “ (espaço) que antecedem e sucedem a cadeia de caracteres “-“ (hífen). Assim, para alinhar o texto a esquerda, deve-se inserir o caractere “:” (dois pontos) antes da cadeia de caracteres “-” (hífen); para alinhar o texto a direita, o caractere “:” (dois pontos) deve ser inserido após a cadeia de caracteres “-“ (hífen); e para que o texto fique centralizado, as duas extremidades da cadeia de caracteres “-“ (hífen) devem conter, ao invés do caractere “ ” (espaço) o caractere “:” (dois pontos).

O Quadro 20 traz um exemplo da marcação feita no texto para obtenção de uma tabela, de modo que, ao visualizar o texto em *Markdown* também é possível prever que a estrutura se trata de uma tabela.

**Quadro 20** - *Markdown:* Sintaxe de formatação *-* Tabelas.

|  |  |
| --- | --- |
| Texto em Markdown | Texto convertido para HTML |
| # Linguagens de programação  |\*ID\*|\*Linguagem | Site\*|  | -- | --------- | ---- |  | 1 | PHP | http://www.php.net |  | 2 | Python | https://www.python.org |  | 3 | Ruby | https://www.ruby-lang.org | | <h1>Linguagens de programação</h1>  <table>  <thead>  <tr>  <th align="center">ID</th>  <th align="left">Linguagem</th>  <th align="right">Site</th>  </tr>  </thead>  <tbody>  <tr>  <td align="center">1</td>  <td align="left">PHP</td>  <td align="right">http://www.php.net</td>  </tr>  <tr>  <td align="center">2</td>  <td align="left">Python</td>  <td align="right">https://www.python.org</td>  </tr>  <tr>  <td align="center">4</td>  <td align="left">Ruby</td>  <td align="right">https://www.ruby-lang.org</td>  </tr>  </tbody>  </table> |

**Fonte:** Autoria própria.

A Figura 37 exibe o resultado renderizado a partir da formatação demonstrada no Quadro 20.

**Figura 37** - *Markdown*: Saída *HTML* renderizada - Tabela.

**Fonte:** Autoria própria.

* 1. PADRONIZAÇÃO DAS ESPECIFICAÇÕES

O tipo de mídia *text/markdown* é registrado pela *RFC-7763*, documento cuja sigla possui o seguinte significado em inglês: *Request For Commments number 7763* (Em português uma tradução condizente é: Requisição de Mudanças número 7763).

Este tipo de documento define um padrão para uso de determinada tecnologia e é emitido pela *Internet Engineering Task Force (IETF)*, um órgão constituído por uma comunidade internacional que trabalha em função da elaboração de padrões abertos para a internet, que são obtidos através de procedimentos que também são abertos, (INTERNET ENGINEERING TASK FORCE (IETF), 2018).

A *RFC-7763* define o referido tipo de mídia para uso com uma família de sintaxe (a linguagem *Markdown* em sua sintaxe original e suas derivações) para “formatação de texto simples, que opcionalmente pode ser convertido em linguagens de marcação formal, como *HTML”* e é complementada pela *RFC*-*7764* que traz orientações acerca da utilização do tipo de mídia *text/Markdown* (LEONARD, 2016, p. 1).

A *RFC-7764* diz que:

“*Markdown* é, especificamente, uma família de sintaxes que se baseiam no trabalho original de John Gruber com contribuições substanciais de Aaron Swartz, lançado em 2004. Desde a sua liberação, uma série de aplicações *web* ou voltadas para a *web* incorporaram o *Markdown* em seus sistemas de entrada de texto, frequentemente com extensões personalizadas” (LEONARD, 2016, p. 4).

A constatação exposta na *RFC-*7764 é de fácil verificação, no entanto a sintaxe original da linguagem não abrange todos os elementos disponíveis na linguagem *HTML* e com o passar do tempo e a constante adoção de *Markdown*, surgiram diversos *softwares* para conversão implementados em várias linguagens de programação, conforme citado na sessão anterior.

Dessa forma, a implementação dada por um desenvolvedor, ou por um grupo de desenvolvedores dentro de um projeto, por vezes destoa das demais implementações, mantendo a compatibilidade apenas com a sintaxe original. Assim, em uma comparação rasa, o usuário que submeter seu texto formatado em *Markdown* ao *parser* *Python* *Markdown*, poderá receber um resultado que não seria igual ao produzido se viesse a submeter o mesmo texto ao *parser* *Kramdown*.

Por essa razão o professor e entusiasta da linguagem *Markdown*, John MacFarlane, publicou a aplicação *web* chamada *Babelmark 2*, utilizada na sessão 3.5.1 no processo de demonstração do uso de *Markdown* para criação de tabelas. A aplicação funciona *online* e permite ao usuário verificar como ficaria a conversão de um texto formatado com *Markdown* após a ação de vários *parsers*.

A aplicação de MacFarlane devolve como resultado de sua execução um quadro, contendo o código HTML gerado por cada um dos conversores, ou a renderização obtida no navegador para o código retornado (MACFARLANE, 2012).

Além dessa aplicação, MacFarlane encabeça um grupo de pessoas usuárias da linguagem que propõe “uma especificação de sintaxe padrão e inequívoca para o *Markdown*, juntamente com um conjunto de testes abrangentes para validar as implementações do *Markdown* contra esta especificação”. A especificação proposta foi intitulada *CommonMark Spec*, (MACFARLANE, GREENSPAN, *et al.*, 2017) e atualmente está na versão 0.28, que foi lançada no dia primeiro de agosto de 2017.

* 1. APLICAÇÃO DA LINGUAGEM MARKDOWN

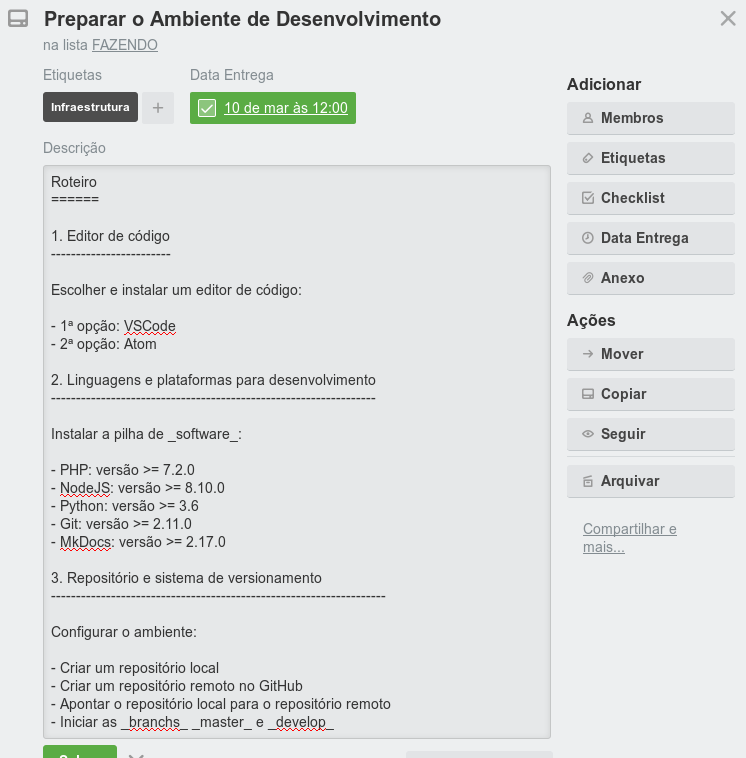
Segundo (GRUBER, 2017) “A sintaxe do *Markdown* destina-se a um propósito: ser usado como um formato para escrever para a web”. Nesse sentido, *Markdown* foi bem recebido por usuários técnicos, desenvolvedores, e por usuários comuns de sistemas diversos tais como: *blogs*, fóruns, *wiki*, gerenciadores de tarefa e outros.

* + 1. Aplicações de uso geral

Um exemplo de aplicação de *Markdown* são os cartões do aplicativo de gestão de projetos e tarefas *Trello*. Nesse aplicativo, que adota a dinâmica da metodologia *Kanban*, tem-se os elementos chamados cartões que representam as tarefas e são organizados em listas dentro de um quadro, sendo que as listas representam o estado de cada tarefa e o quando representa o projeto.

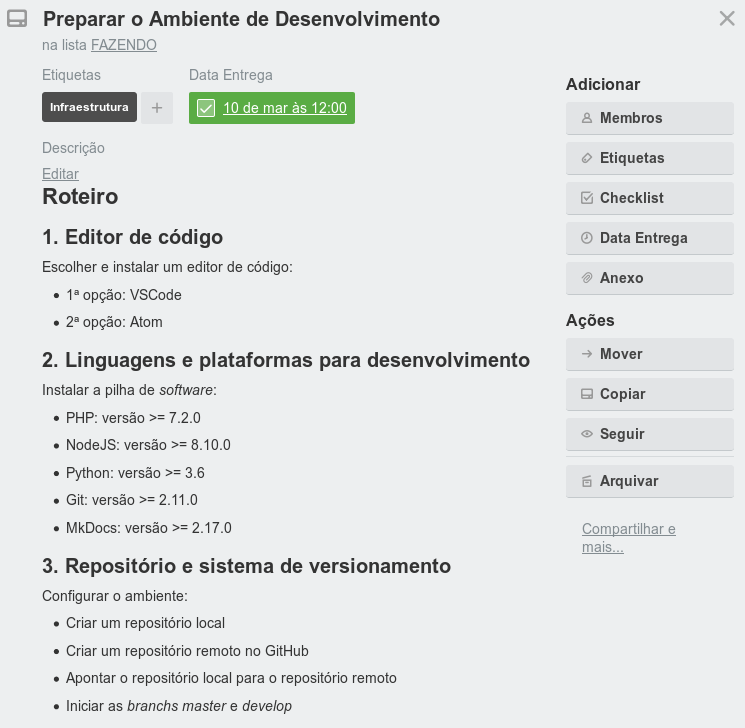
Os cartões do *Trello* dispõem de diversos elementos para inserção de dados sobre a tarefa e entre eles há o campo de descrição textual. Este campo em específico permite a inserção de texto formatado com *Markdown*, proporcionando a elaboração de descrições ricas em detalhes.

A Figura 38 traz um exemplo do uso de *Markdown* em cartões do *Trello*.

**Figura 38** - Sintaxe *Markdown* na formatação do texto de descrição da tarefa no *Trello*.

**Fonte:** Autoria própria.

O resultado da formatação com Markdown no cartão mostrado na Figura 38 pode ser visualizado na Figura 39.

**Figura 39** - Descrição da tarefa feita com *Markdown* renderizada em *HTML* no cartão do *Trello*.

**Fonte:** Autoria própria.

Outro exemplo no qual pode-se aplicar *Markdown* são as interações no *Github* (site para armazenamento e compartilhamento de código), especialmente na funcionalidade de abertura de *issues* (diálogos para reportagem de problemas no código ou esclarecimento de dúvidas em um projeto).

A Figura 40 mostra em detalhes a abertura de uma *issue* para um projeto de *software* armazenado no *GitHub*, de modo que é possível observar o uso da sintaxe *Markdown* para destacar palavras no texto e criar um *link* para um arquivo do projeto.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFigura 40** - Abertura de *issue* no GitHub utilizando *Markdown* para formatar o texto.

**Fonte:** Autoria própria.

O uso da linguagem *Markdown* no *GitHub* é uma prática comum e encorajada, pois propicia a publicação de documentos formatados com detalhes que facilitam o entendimento dos projetos, de eventuais dúvidas e de procedimentos que devem ser adotados por exemplo para instalação e utilização do *software*.

A Figura 41 traz a visualização da *issue*, cuja demonstração foi realizada na Figura 40, após a sua publicação, ou seja, com a formatação *Markdown* já convertida para *HTML* e renderizada.

**Figura 41 -** Visualização da i*ssue* formatada com *Markdown* após a publicação.

Uma imagem contendo captura de tela

Descrição gerada com muito alta confiança

**Fonte:** Autoria própria.

Além das *issues*, o *Github* permite a criação de um arquivo de texto formatado com *Markdown*, que em inglês é chamado de *README.md* (equivalente a “Leia-me” em português). Esse arquivo é colocado no diretório raiz de um projeto de *software* e contém uma descrição geral do repositório e as instruções de uso, instalação, colaboração e outras informações relevantes.

Tal arquivo é automaticamente processado pelo *GitHub*, que exibe o conteúdo de forma renderizada após a listagem de arquivos e diretórios do projeto na página inicial de um repositório.

A Figura 42 traz um recorte de tela, onde observa-se uma demonstração dessa funcionalidade do *GitHub*. A imagem exibe um trecho da página inicial de um repositório armazenado no *GitHub*, especificamente o repositório do *framework* para construção de aplicações *web* na linguagem de programação *PHP* chamado *Laravel*.

**Figura 42** - Recorte de tela da página inicial do repositório Laravel armazenado no Github.

**Uma imagem contendo captura de tela

Descrição gerada com muito alta confiançaFonte:** Autoria própria.

A aplicação da linguagem *Markdown* para construir os arquivos de instruções dos *softwares* disponibilizados em repositórios do *GitHub*, e de outros sites de armazenamento e compartilhamento de código, como o *Gitlab* e o *Bitbucket*, talvez seja o primeiro contato de muitos desenvolvedores com a tecnologia, e o grande caso de sucesso da linguagem *Markdown.*

A partir da simples ideia de se promover a criação de arquivos *README.md* para instruções gerais acerca de um repositório de software, criou-se uma cultura de utilização da linguagem *Markdown* expandida para outras áreas dos sites, como por exemplo a área de *issues,* já citada, e a área de *wiki* (um local para publicação de arquivos como mais detalhes do *software*).

As páginas das áreas de *wiki* dos três sites de armazenamento de repositórios citados, *Github, Gitlab e Bitbucket* são escritas utilizando sintaxe de formatação *Markdown*.

* + 1. Geradores de sites estáticos

Por fim, após toda a explicação e demonstrações do uso de *Markdown* já realizada, resta apenas falar sobre a aplicação em sistemas geradores de sites estáticos.

Pois bem, um site estático se caracteriza pelo fato de que o usuário não insere dados ou interage com a interface de modo a provocar reações maiores do que uma simples troca de páginas, e, no limite, uma pesquisa por conteúdo.

Assim, existem aplicações capazes de converter arquivos escritos em linguagem *Markdown* para sites estáticos compostos por arquivos *HTML* e folhas de estilo *Cascading Style Sheet* (*CSS)*, prontos para serem servidos por um servidor *web*.

O site *StaticGen* (www.staticgen.com) traz uma listagem com dezenas de aplicações para este propósito que estão armazenadas em repositórios do *GitHub*, sendo que, segundo as estatísticas apuradas pelo próprio *GitHub*, o software *Jekyll* é o projeto de gerador de site estáticos que recebeu maior quantidade de estrelas, que é um método de avaliação da popularidade dos repositórios.

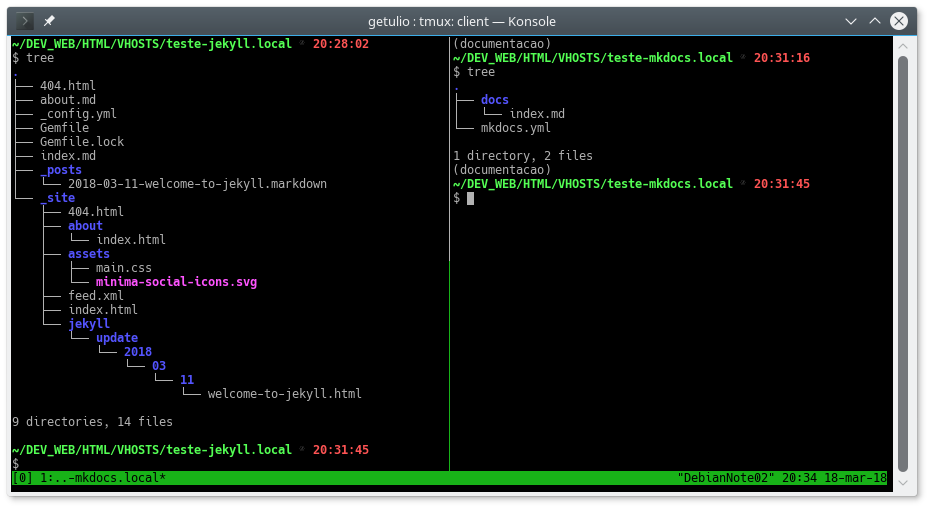
O famoso *Jekyll*, é desenvolvido com a linguagem de programação *Ruby* e se utiliza do *parser* *Kramdown*, que também é escrito em *Ruby*, para converter os arquivos *Markdown* em *HTML*. O *Jekyll*, também é o motor por traz da renderização das páginas no *Github Pages*, um serviço oferecido pelo *GitHub* para servir sites estáticos que apresentam os *softwares* armazenados em repositórios do próprio *GitHub*.

Outro *software* gerador de sites estáticos muito utilizado é o *MkDocs*, escrito com a linguagem de programação *Python*. O *MkDocs* utiliza o *parser* *Python* *Markdown* para converter os arquivos de *Markdown* para *HTML* e, entre suas facetas, está a funcionalidades de envio do site estático, criado em um repositório local (no computador do usuário), para o serviço *GitHub* *Pages*. De maneira similar ao funcionamento do *Jekyll*.

Comparando o *MkDocs* com o *Jekyll*, nota-se que o *Jekyll* possui mais recursos, permitindo até a criação de estruturas complexas de postagens tais como *blogs*. Entretanto o processo de configuração e o tempo de aprendizado da ferramenta podem ser considerados desvantagens do *Jekyll* frente ao *MkDocs*.

A própria instalação das duas aplicações, após concluídas, levam o usuário a concluir que o *MkDocs* possui um ambiente de trabalho menos complexo, o que pode ser observado na Figura 43.

**Figura 43** - Comparação entre a estrutura de um projeto *Jekyll* e um projeto *MkDocs*.

**Fonte:** Autoria própria.

Analisando a Figura 43, do lado direito da imagem tem-se a estrutura de diretórios de uma instalação nova do *Jekyll*, e do lado esquerdo do *MkDocs*.

* + - 1. MkDocs

Por considerar o *MkDocs* um gerador de sites estáticos de mais fácil utilização, além do fato dele atender as necessidades de documentação do projeto que será apresentado no capítulo 5, esta foi a escolha de gerenciador de sites estáticos para demonstração e uso na sequência deste TG.

O processo de instalação do *MkDocs* em um SO *Debian GNU/Linux* será descrito a seguir no Quadro 21.

**Quadro 21** - Processo de instalação do *MkDocs* no SO *Debian GNU/Linux*.

|  |
| --- |
| Instalação do *MkDocs* |
| 1. Fazer o *download* do instalador do gerenciador de pacotes "*pip*". *$ wget https://bootstrap.pypa.io/get-pip.py* 2. Instalar o gerenciador de pacotes "*pip*". *$ sudo python get-pip.py* 3. Instalar o pacote "*virtualenv*". *$ sudo pip install virtualenv* 4. Instalar o pacote "*virtualenvwrapper*". *$ sudo pip install virtualenvwrapper* 5. Editar o arquivo ".*bashrc*" na localizado no diretório "*home*" do usuário atual e adicionando as linhas abaixo no final do arquivo. *# Virtualenvwrapper export WORKON\_HOME=~/.virtualenvs source /usr/local/bin/virtualenvwrapper.sh export PIP\_REQUIRE\_VIRTUALENV=true* 6. Recarregar as configurações do "*bash*". *$ source ~/.bashrc* 7. Criar um diretório para armazenar os arquivos do projeto com "*MkDocs*". *$ mkdir -p ~/teste-mkdocs* 8. Criar um ambiente virtual *Python* para execução do "*MkDocs*" *$ mkvirtualenv docs* 9. Ativar o uso do ambiente virtual *Python*. *$ workon docs\_mkdocs* 10. Instalar o "*MkDocs*". *$ pip install mkdocs* 11. Iniciar um projeto com "*MkDocs*". *$ mkdocs new .*   Os itens numerados representam a descrição das ações que devem ser realizadas (comandos), e, as linhas iniciadas com o caractere “$” (cifrão) representam os comandos que devem ser inseridos, sem o “$”, que apenas indica um ponto de inserção de comandos na CLI em um terminal do SO *Debian GNU/Linux*. |

**Fonte:** Autoria própria.

Com o *MkDocs* instalado, toda a configuração da aplicação é feita alterando os parâmetros contidos no arquivo “*mkdocs.yml*” e, caso necessário, inserindo novos parâmetros. O arquivo “*mkdocs.yml*” é localizado no diretório raiz da instalação, conforme pode ser verificado **Figura 43**.

1. APLICAÇÃO PRÁTICA - PROJETO

Como caso de uso para o estudo elaborado sobre documentação de *software*, foi desenvolvido um projeto de um SI, que nas palavras de (BELMIRO, 2012, p. 6) é definido como “um conjunto de componentes relacionados entre si que coletam (ou recuperam), processam, armazenam e distribuem informações que servem para apoiar a tomada de decisões, a coordenação e o controle de uma organização”.

A seguir serão relatados os processos de Concepção, Planejamento, Execução e Fechamento do projeto.

* 1. CONCEPÇÃO

O projeto foi concebido a partir da constatação de uma necessidade comum a muitas pessoas, a gestão de um orçamento pessoal.

Existem aplicativos no mercado que oferecem esse serviço, alguns de forma gratuita, no entanto existem características do universo contábil, como por exemplo as partidas dobradas, que não são vistos nesses aplicativos, ao menos não de uma maneira fácil de perceber.

Assim, partindo de uma análise dos aplicativos que já existiam e buscando informações com diversas pessoas o projeto foi concebido e modelado

* 1. PLANEJAMENTO
     1. Metodologia

Como metodologia para desenvolvimento do projeto será utilizado o *Kanban*. A escolha se deu por...

* + 1. Ferramentas

Foram definidas...

* 1. EXECUÇÃO

O desenvolvimento do projeto, após a concepção e o planejamento d

* 1. FECHAMENTO

(???)

CONCLUSÃO

Após todo o trabalho...

REFERÊNCIAS

ALEXANDRE, E. D. S. M. Utilização de Markdown para elaboração de TCCs: concepção e experimento da ferramenta Limarka, João Pessoa, 04 mar. 2017. Disponivel em: <http://tede.biblioteca.ufpb.br:8080/handle/tede/9034>. Acesso em: 23 nov. 2017.

ANTONIO, D. Documentação de software – Vilã ou mocinha? **Devmedia**, 2011. Disponivel em: <https://www.devmedia.com.br/documentacao-de-software-vila-ou-mocinha/21795>. Acesso em: 08 jan. 2018.

BARE BONES. BBEdit 12. **Bare Bones**, 2017. Disponivel em: <https://www.barebones.com/products/bbedit/>. Acesso em: 23 nov. 2017.

BECK, K. et al. Manifesto para Desenvolvimento Ágil de Software. **Manifesto for Agile Software Development**, 2001. Disponivel em: <http://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 22 jan. 2018.

BECK, K. et al. Princípios por trás do Manifesto Ágil. **Manifesto for Agile Software Development**, 2001. Disponivel em: <http://agilemanifesto.org/iso/ptbr/principles.html>. Acesso em: 22 jan. 2018.

BELMIRO, N. J. **Sistemas de Informação**. São Paulo: Pearson Education do Brasil, 2012.

BLANC, S. John Gruber: A Mix of the Technical, the Artful, the Thoughtful, and the Absurd. **Shawn Blanc**, 18 fev. 2008. Disponivel em: <http://shawnblanc.net/2008/02/interview-john-gruber/>. Acesso em: 05 nov. 2017.

CODE.ORG. About US | Code.org. **Code.org**, 2017. Disponivel em: <https://code.org/about>. Acesso em: 03 nov. 2017.

FERREIRA, E.; EIS, D. HTML5 Curso W3C Escritório Brasil. **W3C Brasil**, 2011. Disponivel em: <http://www.w3c.br/Cursos/CursoHTML5>. Acesso em: 23 nov. 2017.

FILHO, W. D. P. P. **Engenharia de Software - Fundamentos, Métodos e Padrões**. 2. ed. ed. Rio de Janeiro: LTC, 2003.

FORTIN, M. PHP Markdown. **Michel Fortin**, 14 jan. 2018. Disponivel em: <https://michelf.ca/projects/php-markdown/>. Acesso em: 16 mar. 2018.

FRAMEWORKS. **Projeto de Software Orientado a Objeto - Programa**. Disponivel em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 13 fev. 2018.

FURTADO, K. Metodologia ágil: documentação para projetos ágeis. **Devmedia**, 2016. Disponivel em: <https://www.devmedia.com.br/metodologia-agil-documentacao-para-projetos-ageis/37577>. Acesso em: 03 nov. 2017.

GRUBER, J. Dive Into Markdown. **Daring Fireball**, 19 mar. 2004. Disponivel em: <https://daringfireball.net/2004/03/dive\_into\_markdown>. Acesso em: 09 nov. 2017.

GRUBER, J. Introducing Markdown. **Daring Fireball**, 15 mar. 2004. Disponivel em: <https://daringfireball.net/2004/03/introducing\_markdown>. Acesso em: 9 nov. 2017.

GRUBER, J. Markdown. **Daring Fireball**, 2017. Disponivel em: <https://daringfireball.net/projects/markdown/>. Acesso em: 04 nov. 2017.

GRUBER, J. Markdown: Basics. **Daring Fireball**, 2017. Disponivel em: <https://daringfireball.net/projects/markdown/basics>. Acesso em: 12 nov. 2017.

GRUBER, J. Markdown: Dingus. **Daring Fireball**, 2017. Disponivel em: <https://daringfireball.net/projects/markdown/dingus>. Acesso em: 12 nov. 2017.

GRUBER, J. Markdown: Syntax. **Daring Fireball**, 2017. Disponivel em: <https://daringfireball.net/projects/markdown/syntax>. Acesso em: 12 nov. 2017.

GUANABARA, G. Curso Python #01 - Seja um Programador. **YouTube**, Rio de Janeiro, 3 abr. 2017. Disponivel em: <https://www.youtube.com/watch?v=S9uPNppGsGo&list=PLHz\_AreHm4dlKP6QQCekuIPky1CiwmdI6>. Acesso em: 2 nov. 2017.

HEUSER, C. A. **Projeto de banco de dados**. 6 Ed. ed. Porto Alegre: Bookman, 2009.

INKSCAPE. Tutoriais em Texto. **Inkscape**, 2018. Disponivel em: <https://inkscape.org/pt-br/aprender/tutoriais/>. Acesso em: 11 mar. 2018.

INTERNET ENGINEERING TASK FORCE (IETF). About. **Internet Engineering Task Force (IETF)**, 2018. Disponivel em: <https://www.ietf.org/about/>. Acesso em: 16 mar. 2018.

LEITNER, T. kramdown. **kramdown**, 2017. Disponivel em: <https://kramdown.gettalong.org/>. Acesso em: 16 mar. 2018.

LEONARD, S. Guidance on Markdown: Design Philosophies, Stability Strategies, and Select Registration. **Internet Engineering Task Force (IETF)**, mar. 2016. Disponivel em: <https://tools.ietf.org/html/rfc7764>. Acesso em: 16 mar. 2018.

LEONARD, S. RFC-7763 The text/markdown Media Type. **Internet Engineering Task Force (IETF)**, mar. 2016. Disponivel em: <https://tools.ietf.org/html/rfc7763>. Acesso em: 16 mar. 2018.

LYNX. Lynx. **Lynx**, 2017. Disponivel em: <https://lynx.browser.org/>. Acesso em: 23 nov. 2017.

MACFARLANE, J. Babelmark 2. **John MacFarlane**, 2012. Disponivel em: <http://johnmacfarlane.net/babelmark2/>. Acesso em: 16 mar. 2018.

MACFARLANE, J. About Pandoc. **Pandoc - a universal document converter**, 2017. Disponivel em: <http://pandoc.org>. Acesso em: 16 mar. 2018.

MACFARLANE, J. et al. CommonMark. **CommonMark**, 2017. Disponivel em: <http://commonmark.org/>. Acesso em: 05 nov. 2017.

MALHERBI, E. Prototipação de Sistemas utilizando a Ferramenta Balsamiq Mockup. **Devmedia**, 2013. Disponivel em: <https://www.devmedia.com.br/prototipacao-de-sistemas-utilizando-a-ferramenta-balsamiq-mockup/27232>. Acesso em: 06 nov. 2017.

MARTINS, R. Kanban: 4 passos para implementar em uma equipe. **Devmedia**. Disponivel em: <https://www.devmedia.com.br/kanban-4-passos-para-implementar-em-uma-equipe/30218>. Acesso em: 13 fev. 2018.

MDN WEB DOC - MOZILLA. Tag. **MDN Web Doc - Mozilla**, 03 ago. 2015. Disponivel em: <https://developer.mozilla.org/en-US/docs/Glossary/Tag>. Acesso em: 22 nov. 2017.

MDN WEB DOC - MOZILLA. code. **MDN Web Doc - Mozilla**, 25 out. 2017. Disponivel em: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/code>. Acesso em: 23 nov. 2017.

MDN WEB DOC - MOZILLA. em. **MDN Web Doc - Mozilla**, 19 out. 2017. Disponivel em: <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/em>. Acesso em: 20 nov. 2017.

MDN WEB DOC - MOZILLA. strong. **MDN Web Doc - Mozilla**, 04 ago. 2017. Disponivel em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/strong>. Acesso em: 20 nov. 2017.

MOVABLETYPE.ORG. What is Movable Type? **MovableType.Org**, 2017. Disponivel em: <https://www.movabletype.org/>. Acesso em: 23 nov. 2017.

OBJECT MANAGEMENT GROUP. **OMG Unified Modeling Language (OMG UML)**. Version 2.5.1. ed. Needham: [s.n.], 2017. Disponivel em: <https://www.omg.org/spec/UML/2.5.1>.

OBJECT MANAGEMENT GROUP, INC. (OMG). **Business Processs Model and Notation (BPMN) - Version 2.0**. [S.l.]: [s.n.], 2011. Disponivel em: <http://www.omg.org/spec/BPMN/2.0>.

PRESSMAN, R. S. **Sistemas de Informação - Uma Abordagem Profissional**. 7. ed. ed. Porto Alegre: AMGH, 2011.

ROBERT, K. S. perlintro. **perldoc.perl.org Pear Programming Documentation**, 2017. Disponivel em: <http://perldoc.perl.org/perlintro.html>. Acesso em: 23 nov. 2017.

SCHWBER, K.; SUTHERLAND, J. **Guia do Scrum**. [S.l.]: [s.n.], 2017. Disponivel em: <http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-Brazilian.pdf>. Acesso em: 13 fev. 2018.

SENE, R. P. D. Processo, qualidade e métricas de desenvolvimento de software. **Engenharia de Software Magazine**, Rio de Janeiro, n. 55, p. 54, 2012. Disponivel em: <https://www.devmedia.com.br/revista-engenharia-de-software-magazine-55/26939>.

SIGNIFICADOS. Significado de Blog. **Significados**, 2017. Disponivel em: <https://www.significados.com.br/blog/>. Acesso em: 23 nov. 2017.

SMITH, L. Best Practices for Plain-Text Emails + A Look at Why They’re Important. **litmus**, 15 out. 2014. Disponivel em: <https://litmus.com/blog/best-practices-for-plain-text-emails-a-look-at-why-theyre-important>. Acesso em: 09 nov. 2017.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. ed. São Paulo: Pearson, 2011.

SOUZA, V. R. D.; VALE, R. C.; ARAÚJO, M. A. P. Prototipação no Desenvolvimento de Software. **Engenharia de Software**, Rio de Janeiro, n. 17, p. 66, 2008. Disponivel em: <https://www.devmedia.com.br/revista-engenharia-de-software-17/14495>.

THE PYTHON-MARKDOWN PROJECT. Python-Markdown. **Python-Markdown**, 2017. Disponivel em: <https://python-markdown.github.io/>. Acesso em: 16 mar. 2018.

VUE.JS. Vue.js - Guia 2.x. **Vue.js**, 2018. Disponivel em: <https://br.vuejs.org/v2/guide/>. Acesso em: 11 mar. 2018.

W3C. XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition). **W3C**, 26 jan. 2000. Disponivel em: <https://www.w3.org/TR/xhtml1/>. Acesso em: 23 nov. 2017.

1. ***HTML*:** Linguagem de marcação de texto estruturado para publicação de conteúdo (texto, imagem, vídeo e áudio) na *web* (FERREIRA e EIS, 2011). [↑](#footnote-ref-1)
2. ***Weblog*:** O termo é resultante da justaposição das palavras da língua inglesa *web* e *log*. *Web* neste contexto representa rede (*internet*) enquanto que *log* é utilizado para designar o registro de atividade regular (SIGNIFICADOS, 2017). [↑](#footnote-ref-2)
3. ***Tag*: R**efere-se a definição atribuída aos elementos sintáticos do texto, utilizados na linguagem *HTML que são* responsáveis por delimitar o início e o fim da construção de um elemento da estrutura de um arquivo *HTML* (MDN WEB DOC - MOZILLA, 2015). [↑](#footnote-ref-3)
4. ***BBEdit*:** Editor de textos *HTML* feito para o sistema operacional *Macintosh* da empresa *Apple* (BARE BONES, 2017). [↑](#footnote-ref-4)
5. ***Login*:** Autenticação em um sistema. [↑](#footnote-ref-5)
6. ***MT*:** Abreviação para *Movable Type*, que é um *software* para criação e gerenciamento de conteúdo para *web*. Em inglês *CMS (Content Management System)* (MOVABLETYPE.ORG, 2017). [↑](#footnote-ref-6)
7. ***Perl*:** “Linguagem de programação de uso geral originalmente desenvolvida para manipulação de texto e agora usada para uma ampla gama de tarefas, incluindo administração de sistema, desenvolvimento web, programação de rede, desenvolvimento de GUI e muito mais” (ROBERT, 2017). [↑](#footnote-ref-7)
8. ***XHTML*:** *(Extensible Hiper Text Markup Language)* é um formato que estende a marcação *HTML* permitindo o tratamento dos documentos como *XML (Extensible Markup Language)* (W3C, 2000). [↑](#footnote-ref-8)
9. **Válido:** Refere-se ao emprego correto das marcações *(tags)* *HTML e XHTML* no documento. [↑](#footnote-ref-9)
10. [↑](#footnote-ref-10)
11. [↑](#footnote-ref-11)
12. [↑](#footnote-ref-12)