UNIVERSIdade FEEVALE

LUCAS EDUARDO HARTZ

UMA PROPOSTA DE SISTEMA DE RECOMENDAÇÃO PARA CONSTRUÇÃO DE UM PROCESSO DE SOFTWARE

Novo Hamburgo

2017

lucas eduardo hartz

UMA PROPOSTA DE SISTEMA DE RECOMENDAÇÃO PARA CONSTRUÇÃO DE UM PROCESSO DE SOFTWARE

Trabalho de Conclusão de Curso

apresentado como requisito parcial

à obtenção do grau de Bacharel em

Sistemas de Informação pela

Universidade Feevale.

Orientadora: Adriana Neves dos Reis

Novo Hamburgo

2017

Agradecimentos

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial:

A Thais, minha namorada, que passou horas me auxiliando e me motivando, aos meus pais, amigos e às pessoas que convivem comigo diariamente, minha gratidão, pelo apoio emocional nos períodos mais difíceis do trabalho.

Em especial, a minha orientadora Adriana, pela sua dedicação, motivação e auxílio durante todo o desenvolvimento deste trabalho.

Resumo

No processo de criação de software existem diversas abordagens para definir sua forma de condução. Assim como as linguagens de programação que estão cada vez mais sofisticadas, sem deixar a praticidade de lado, os métodos de desenvolvimento de software estão constantemente se atualizando. Isso na busca por novas formas de fazer o mesmo produto com menos tempo, utilizando todo o potencial de uma equipe, com chances de reduzir o custo final, e, por consequência, ganhando competitividade no mercado de softwares. Em relação aos processos de Engenharia de Software, duas abordagens são encontradas na literatura: as metodologias ágeis e os modelos tradicionais. As metodologias ágeis são um conjunto de boas práticas, que focam em desenvolvimento ágil, qualidade de software e satisfação do cliente, as quais foram desenvolvidas para agilizar os processos e a qualidade na gestão de softwares. As metodologias tradicionais, por sua vez, focam na documentação do software como forma de comunicação entre a equipe nos diferentes estágios do processo de desenvolvimento. Na forma de práticas ou modelos, sabe-se que na realidade é possível juntar as estratégias mais eficientes de cada abordagem, e moldar o nosso próprio processo. O objetivo desse trabalho foi construir um *framework* para auxiliar os usuários na recomendação de uma técnica apropriada para a demanda da empresa na construção de um processo de software. Realizou-se um estudo piloto empregando a ferramenta com os alunos da disciplina de Engenharia de Software. Os alunos responderam um questionário referente as perguntas abordadas dentro da ferramenta, a fim de avaliar a eficiência do *framework*. Ao final deste trabalho, conclui-se que o *framework* está apto para auxiliar os usuários na busca de um método apropriado.

Palavras-chave: Processos de Software; Engenharia de Software; Metodologia Ágil; Metodologias Tradicionais; *Framework*.

Abstract

In the process of creating software there are several approaches to define your driving style. As well as programming languages ​​that are increasingly sophisticated, but without leaving practicality aside, software development methods are constantly being updated. This is in search of new ways to do the same product with less time, using the full potential of a team, with chances of reducing the final cost, and, consequently, gaining competitiveness in the software market. In relation to Software Engineering processes, two approaches are found in the literature: agile methodologies and traditional models. Agile methodologies are a set of good practices, focused on agile development, software quality and customer satisfaction, which were developed to streamline processes and quality in software management. Traditional methodologies, in turn, focus on software documentation as a form of communication between the team at different stages of the development process. In the form of practices or models, it is known that in reality it is possible to combine the most efficient strategies of each approach, and to shape our own method. The objective of this work was to build a framework to recomend users in defining a technique more appropriate to the company's demand in the construction of a software process. A pilot study was carried out using the tool with the students of the Software Engineering discipline. The students answered a questionnaire regarding the questions addressed within the tool, in order to evaluate the efficiency of the framework. At the end of this work, it is concluded that the framework is able to assist users in the search for an appropriate method.

Key-words: Software Processes; Software Engineering; Agile Methodology; Traditional Methodologies; Framework.

LISTA DE FIGURAS

Figura 1 - Camadas da Engenharia de Software.....................................................................16

Figura 2 - Modelo Cascata.........................................................................................................20

Figura 3 - Ciclo do modelo de prototipagem............................................................................23

Figura 4 - Modelo Espiral..........................................................................................................25

Figura 5 - Fluxo do processo Scrum..........................................................................................33

Figura 6 - Diagrama de caso de uso em UML..........................................................................41

Figura 7 - Página principal do *framework*................................................................................56

Figura 8 - Planilha de resultados..............................................................................................63

Figura 9 - Pesos de requisitos de requisitos para produtos novos..........................................65

LISTA DE QUADRO

Quadro 1 - Vantagens e desvantagens do Modelo Cascata......................................................22

Quadro 2 - Vantagens e desvantagens do *Extreme Programming*...........................................31

Quadro 3 - Técnicas de UML......................................................................................................44

Quadro 4 - Compreensão dos níveis.........*..*...............................................................................55

Quadro 5 - Perguntas para produtos novos..............................................................................59

Quadro 6 - Perguntas para atualização.....................................................................................60

Quadro 7 - Pesos de produtos novos..........................................................................................66

Quadro 8 - Pesos para atualização.............................................................................................70

Lista de Abreviaturas e Siglas

|  |  |
| --- | --- |
| DD | Dicionário de Dados |
| DER | Diagrama de Entidade-Ralacionamento |
| DFD | Diagrama de Fluxo de Dados |
| JAD | *Joint Application Development* |
| MED | Modelo de Entidade de Relacionamento |
| MDAMPS.BR | *Model-Driven Architecture*Melhoria de Processos do Software Brasileiro |
| RUP | *Rational Unified Process* |
| SI | Sistemas de Informação |
| UML | *Unified Modeling Language* |
| XP | *Extreme Programming*  |

Sumário

[Resumo 4](#_Toc486963133)

[Abstract 5](#_Toc486963134)

[LISTA DE FIGURAS 6](#_Toc486963135)

[LISTA DE QUADRO 7](#_Toc486963136)

[Introdução 13](#_Toc486963137)

[1 Software 16](#_Toc486963138)

[1.1 Engenharia de Software 16](#_Toc486963139)

[1.2 Processos de Software 17](#_Toc486963140)

[2 Modelos de processos 20](#_Toc486963141)

[2.1 Métodos tradicionais 20](#_Toc486963142)

[2.1.1 Modelo Cascata 20](#_Toc486963143)

[2.1.2 Modelo de Prototipação 23](#_Toc486963144)

[2.1.3 Modelo Espiral 25](#_Toc486963145)

[2.1.4 RUP – Rational Unified Process 27](#_Toc486963146)

[2.1.5 Considerações sobre métodos Tradicionais. 30](#_Toc486963147)

[2.2 Métodos Ágeis 30](#_Toc486963148)

[2.2.1 Extreme Programming (XP) 30](#_Toc486963149)

[2.2.2 Scrum 32](#_Toc486963150)

[2.2.3 Considerações sobre métodos ágeis 35](#_Toc486963151)

[3 Análise do ciclo de vida 36](#_Toc486963152)

[3.1 Requisitos, Especificações 36](#_Toc486963153)

[3.1.1 Entrevistas 36](#_Toc486963154)

[3.1.2 Workshop 37](#_Toc486963155)

[3.1.3 Brainstorming 37](#_Toc486963156)

[3.1.4 Questionário 38](#_Toc486963157)

[3.1.5 Observação 38](#_Toc486963158)

[3.1.5.1 Etnografia 39](#_Toc486963159)

[3.1.6 JAD 39](#_Toc486963160)

[3.1.7 Prototipagem 40](#_Toc486963161)

[3.1.8 Casos de Uso 41](#_Toc486963162)

[3.1.9 Histórias de usuário 42](#_Toc486963163)

[3.1.10 Considerações sobre requisitos 43](#_Toc486963164)

[3.2 Análise 43](#_Toc486963165)

[3.2.1 Análise orientada a objetos 43](#_Toc486963166)

[3.2.1.1 UML 44](#_Toc486963167)

[3.2.2 Análise Estruturada 46](#_Toc486963168)

[3.2.2.1 Diagrama de Fluxo de Dados 46](#_Toc486963169)

[3.2.2.2 Dicionário de Dados 46](#_Toc486963170)

[3.2.2.3 Diagrama Entidade-Relacionamento (DER) 46](#_Toc486963171)

[3.2.3 Considerações sobre análise 47](#_Toc486963172)

[3.3 Design de software 47](#_Toc486963173)

[3.3.1 Modelo de Entidade Relacional 47](#_Toc486963174)

[3.3.2 Model-Driven Architecture (MDA) 48](#_Toc486963175)

[3.3.3 Design Pattern 48](#_Toc486963176)

[3.3.4 Refatoração 48](#_Toc486963177)

[3.3.5 Considerações sobre Design de Software 48](#_Toc486963178)

[3.4 Construção 49](#_Toc486963179)

[3.4.1 Programação em Par 49](#_Toc486963180)

[3.5 Testes 49](#_Toc486963181)

[3.5.1 Teste de Unidade 49](#_Toc486963182)

[3.5.1.1 Teste de Unidade para Software Orientado a Objeto 50](#_Toc486963183)

[3.5.2 Teste de Integração 50](#_Toc486963184)

[3.5.2.1 Teste de Fumaça 51](#_Toc486963185)

[3.5.3 Teste de Validação 51](#_Toc486963186)

[3.5.4 Teste de Sistema 51](#_Toc486963187)

[3.5.4.1 Teste de Recuperação 52](#_Toc486963188)

[3.5.4.2 Testes de Segurança 52](#_Toc486963189)

[3.5.4.3 Teste por Esforço 52](#_Toc486963190)

[3.5.4.4 Teste de Desempenho 52](#_Toc486963191)

[3.5.4.5 Teste de Disponibilidade 52](#_Toc486963192)

[3.5.5 Teste Caixa-Branca 52](#_Toc486963193)

[3.5.6 Teste Caixa-Preta 53](#_Toc486963194)

[3.5.7 Considerações sobre testes 53](#_Toc486963195)

[3.6 Manutenção 53](#_Toc486963196)

[3.6.1 Manutenção Corretiva 53](#_Toc486963197)

[3.6.2 Manutenção Adaptativa 54](#_Toc486963198)

[3.6.3 Manutenção Perfectiva 54](#_Toc486963199)

[3.6.4 Manutenção Preventiva 54](#_Toc486963200)

[3.6.5 Considerações sobre manutenção 54](#_Toc486963201)

[4 construção do FRAMEWORK 55](#_Toc486963202)

[4.1 Página principal do framework 55](#_Toc486963203)

[4.2 Perguntas ao usuário do framework 57](#_Toc486963204)

[4.3 Tabela de resultados 62](#_Toc486963205)

[4.3.1 Pesos das perguntas do framework 65](#_Toc486963206)

[4.4 Protocolo do usuário 76](#_Toc486963207)

[4.5 Considerações sobre o framework 76](#_Toc486963208)

[5 avaliação do Framework 77](#_Toc486963209)

[5.1 Estudo piloto 77](#_Toc486963210)

[5.1.1 Análise do ambiente 77](#_Toc486963211)

[5.1.2 Estudo individual 78](#_Toc486963212)

[5.1.3 Estudo em grupo 78](#_Toc486963213)

[5.1.4 Estudo de caso 79](#_Toc486963214)

[5.2 Questionário 79](#_Toc486963215)

[5.2.1 Avaliação das perguntas 81](#_Toc486963216)

[5.3 Considerações sobre a avaliação do framework 83](#_Toc486963217)

[CONCLUSÃO 84](#_Toc486963218)

[Referências Bibliográficas 86](#_Toc486963219)

Introdução

 Segundo Pressman (2016), a motivação de criar um software vem basicamente da necessidade. Seguindo essa ideia podemos dizer que o software é a percepção de uma necessidade, logo, é importante satisfazer a expectativa das pessoas que possuem interesse no seu uso, entregando um programa funcional, dentro do prazo estipulado e que atenda todas suas exigências especificadas. Para facilitar esse procedimento, foram criados modelos de processos de software, que servem de base para definição de processos propostos de gestão da produção de programas.

 Sommerville (2011) afirma que se tratando de processos desoftware não existe um processo “ideal”, devido às diversidades das equipes dentro das organizações. Desta forma, os processos necessitam de melhorarias ou reestruturação para usufruir o máximo do potencial de cada colaborador, bem como das características específicas do projeto em desenvolvimento.

 Conforme Pressman (2016),

“O papel desempenhado pelo software tem passado por grandes mudanças ao longo dos últimos cinquenta anos. Aperfeiçoamentos significativos no desempenho do hardware, mudanças profundas nas arquiteturas computacionais, vasto aumento na capacidade de memória e armazenamento, e uma ampla variedade de exóticas opções de entrada e saída, tudo isso resultou em sistemas computacionais mais sofisticados e complexos. Sofisticação e complexidade podem produzir resultados impressionantes quando um sistema é bem-sucedido, porém, também podem trazer enormes problemas para aqueles que precisam desenvolver sistemas robustos. Atualmente, uma enorme indústria de software tornou-se fator dominante nas economias do mundo industrializado. Equipes de especialistas em software, cada qual concentrando-se numa parte da tecnologia necessária para distribuir uma aplicação complexa, substituíram o programador solitário de antigamente. Ainda assim, as questões levantadas por esse programador solitário continuam as mesmas feitas hoje, quando os modernos sistemas computacionais são desenvolvidos. ”

 As metodologias tradicionais foram as primeiras a surgirem, proporcionando uma considerável contribuição quanto à estrutura utilizável no trabalho de Engenharia de Software e fornecendo uma espécie de itinerário razoavelmente eficaz para equipes de softwares. Essas são conhecidas por possuírem um grande volume de documentação, os quais servem como mecanismo de comunicação entre os diferentes papéis atuantes na construção do software (PRESSMAN, 2016).

 O grande problema das metodologias tradicionais é seu grande volume de documentação, o que demanda muito tempo para se criar, utilizando o tempo de um funcionário para criar documentação, muitas vezes desnecessária.

 Em fevereiro de 2001, renomeados programadores se reuniram para criticar os métodos tradicionais de desenvolvimento de software, argumentando que eles eram lentos e não lidavam bem com mudanças de escopo no projeto. No debate sobre como poderiam ganhar agilidade em desenvolvimento de software, que foi construído o Manifesto Ágil, embora o desenvolvimento ágil já estivesse em evidência, foi nessa data que se consolidou um “movimento”. Nesta oportunidade, foram criados os pilares do Manifesto Ágil, sendo eles: indivíduos e interação entre eles mais que processos e ferramentas, software em funcionamento mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos, e responder a mudanças mais que seguir um plano (MANIFESTO ÁGIL, 2001).

 Assim, os métodos ágeis se desenvolveram em um esforço para sanar fraquezas reais e visíveis da Engenharia de Software convencional. O desenvolvimento ágil tem diversos benefícios, porém não é indicado a todos os projetos (PRESSMAN, 2011).

 O problema está na escolha de qual metodologia é a mais indicada para a equipe e para o escopo do projeto. Muitas empresas investem na troca da metodologia já empregada na empresa, sem saber se essa mudança trará benefícios. A intenção do trabalho é mapear os modelos existentes na literatura, catalogá-los e montar uma ferramenta que busque o método mais apropriado para o desenvolvimento do projeto.

 A motivação deste trabalho é mapear elementos dos processos de software existentes, questionando se é possível agregar valor à Engenharia de Software se combinarmos práticas já conhecidas de ambas metodologias, com intuito de montar um processo que combine as melhores técnicas das metodologias ágeis e tradicionais em um *framework* com a hipótese de auxiliar ou não a equipe na escolha da melhor maneira de conduzir um projeto de desenvolvimento de software.

 O trabalho está organizado da seguinte forma: o capítulo 1 introduz o assunto, falando sobre o que é um software, qual a necessidade de existir a Engenharia de Software e o que é um processo de software*.* O capítulo 2 apresenta os modelos de processos existentes na literatura, métodos tradicionais e métodos ágeis. O terceiro capítulo mostra os ciclos de vida de um software, que são basicamente Requisitos, Análise, *Design*, Construção, Testes e Manutenção.

 O quarto capítulo aborda sobre como foi construído o *framework*, sua estrutura e sua finalidade de elaboração. O quinto capítulo abordará sobre o estudo piloto trará os resultados obtidos a partir do estudo piloto. Após, é realizada a conclusão do trabalho.

# Software

O software distribui o produto mais importante da nossa era, a informação. Atualmente é um produto e o veículo para distribuir um produto. Como produto, fornece o potencial computacional representado pelo hardware ou, de forma mais abrangente, por uma rede de computadores que podem ser acessados por hardware local. O software é um transformador de informações, ou seja, produz, gerencia, adquire, modifica, exibe ou transmite informações que podem ser simples ou complexas. Como veículo de distribuição do produto, atua como base para o controle do computador (sistemas operacionais), a comunicação de informações (redes) e a criação e o controle de outros programas (ferramentas de software e ambientes) (PRESSMAN, 2016).

Este capítulo apresenta o conceito de software, qual a necessidade de sua existência, apresentar a engenharia de software, quais técnicas existentes para a construção de um programa e o que são processos de software.

## Engenharia de Software

 As organizações têm buscado por melhoria contínua de processos que satisfaçam suas necessidades e as tornem mais competitivas frente ao mercado. Desta forma o desenvolvimento de software torna-se uma atividade de grande importância. A adoção de metodologias para padronização e gerência nos processos de desenvolvimento de software com o intuito de buscar qualidade nos processos e produtos de software proporciona ganho de competitividade no mercado, pois garante serviços ou produtos de forma rápida, com custos mais baixos e com mais qualidade e segurança (OLIVEIRA, 2012).

 O software está incorporado em praticamente todos os aspectos de nossas vidas e, consequentemente, tem crescido significativamente o número de pessoas interessadas nos recursos e nas funções oferecidas por uma determinada aplicação. Quando uma aplicação ou um sistema embutido estão para ser desenvolvidos deve-se compreender o problema antes de desenvolver uma solução de software (PRESSMAN, 2016).

 Os requisitos de tecnologia de informação estão se tornando cada vez mais complexos. Atualmente, grandes equipes desenvolvem programas de computador que antigamente eram desenvolvidos por um único indivíduo. Um software sofisticado está incorporado em tudo, de produtos eletrônicos de consumo a equipamentos médicos e sistemas de armamentos. A complexidade desses novos produtos e sistemas baseados em computadores demanda uma maior atenção para com as interações de todos os elementos do sistema. Desta forma, projetar tornou-se uma atividade fundamental (PRESSMAN, 2016).

 Segundo Sommerville (2011), a Engenharia de Software é uma tecnologia em camadas, conforme apresenta a Figura 1, em qual quer abordagem de engenharia deve estar fundamentada em comprometimento organizacional com a qualidade.

Figura : Camadas da Engenharia de Software



Fonte: PRESSMAN, 2016.

 Segundo PRESSMAN (2016) qualquer abordagem de engenharia, inclusive Engenharia de Software, deve estar fundamentada em um comprometimento organizacional com a qualidade. A gestão da qualidade promove uma cultura de aperfeiçoamento contínuo de processos no qual leva ao desenvolvimento de abordagens cada vez mais efetivas na engenharia de software, ou seja, o que sustenta a engenharia de software é o foco na qualidade.

 Os métodos da Engenharia de Software fornecem as informações técnicas para desenvolver o software. Esses métodos envolvem uma ampla gama de tarefas, incluindo a comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Baseiam-se em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descritivas (PRESSMAN, 2016).

## Processos de Software

 Um Processo de Software é um conjunto de atividades que levam à produção de um produto de software, sendo este adaptável, possibilitando às pessoas da equipe de software realizar o trabalho de selecionar e escolher o conjunto apropriado de ações e tarefas. A intenção é a de sempre entregar software dentro do prazo e com qualidade suficiente para satisfazer àqueles que patrocinaram sua criação e àqueles que irão utilizá-lo (PRESSMAN, 2016; SOMMERVILLE, 2011).

 Existem muitos modelos de processos de software diferentes, porém um conjunto de atividades de apoio são aplicáveis a todos eles. Uma metodologia de processo genérica para Engenharia de Software compreende cinco atividades (PRESSMAN, 2016):

* **Comunicação**: comunicar-se e colaborar com o cliente, ou seja, compreender os objetivos das partes interessadas para com o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software.
* **Planejamento**: realizar um planejamento ajuda a guiar a equipe de software. Neste planejamento deve definir o trabalho de engenharia de software, descrevendo as tarefas técnicas a ser conduzidas, os riscos prováveis, os recursos que serão necessários, os produtos resultantes a ser produzidos e um cronograma de trabalho.
* **Modelagem**: cria-se um esboço do software, criando modelos para melhor atender as necessidades do software e o projeto que irá atender as essas necessidades. Se necessário, refina-se o esboço com mais detalhes, numa tentativa de compreender melhor o problema e como resolvê-lo.
* **Construção**: construir o código (manual ou automatizada) e testar o que for necessário para revelar erros na codificação.
* **Emprego**: Entregar o software (como uma entidade completa ou como um incremento parcialmente efetivado) ao cliente, que avalia o produto entregue e fornece *feedback*, baseado na avaliação.

 As fases metodológicas do processo de engenharia de software são complementadas por uma série de atividades de apoio que em geral são aplicadas ao longo de um projeto, ajudando a equipe a gerenciar, a controlar o progresso, a qualidade, as mudanças e o risco. As atividades de apoio típicas são (PRESSMAN, 2016):

* **Controle e acompanhamento do projeto**: avaliar o progresso em relação ao plano do projeto e toma as medidas necessárias para cumprir o cronograma.
* **Administração de riscos**: avaliar riscos que possam afetar o resultado ou a qualidade do produto/projeto.
* **Garantia da qualidade de software**: definir e conduzir as atividades que garantem a qualidade do software.
* **Revisões técnicas**: avaliar artefatos da engenharia de software, tentando identificar e eliminar erros antes de passar para a próxima etapa.
* **Medição**: definir e coletar medidas (do processo, do projeto e do produto). Auxilia na entrega do software de acordo com os requisitos.
* **Gerenciamento da configuração de software**: gerenciar os efeitos das mudanças ao longo do processo.
* **Gerenciamento da reusabilidade**: definir critérios para o reuso de artefatos (inclusive componentes de software) e estabelece mecanismos para a obtenção de componentes reutilizáveis.
* **Preparo e produção de artefatos de software**: englobar as atividades necessárias para criar artefatos como, por exemplo, modelos, documentos, logs, formulários e listas.

 Segundo PRESSMAN (2016) o processo de Engenharia de Software não é rígido nem deve ser seguido à risca. Deve ser ágil e adaptável (ao problema, ao projeto, à equipe e à cultura organizacional), portanto, o processo adotado para um determinado projeto pode ser muito diferente daquele adotado para outro.

# Modelos de processos

 Um Modelo de Processo de *Software* é uma representação simplificada de um processo de software. Cada modelo representa uma perspectiva particular de processo, e, portanto, fornece informações parciais sobre ele (SOMMERVILLE, 2011). Este capítulo apresenta os métodos e modelos já existentes. Na literatura, encontra-se mais facilmente os métodos Tradicionais e os Ágeis.

## Métodos tradicionais

 Nos métodos tradicionais, quando uma equipe pretende desenvolver um software, eles utilizam um plano de desenvolvimento dirigido. Este tipo de desenvolvimento é caracterizado por etapas fechadas, onde reúnem-se todos os requisitos do cliente necessários para o projeto e depois realizam-se a análise do problema. Em seguida, toda a aplicação é concebida antes da primeira linha de código que nunca foi escrito. Uma das metodologias amplamente utilizadas é o Modelo Cascata (BLANKENSHIP; BUSSA; MILLETT, 2011).

### Modelo Cascata

 O Modelo Cascata proposto por Winston Royce em 1970 (PRESSMAN, 2016) é o modelo clássico de engenharia de software e um dos mais antigos. Este modelo enfatiza o planejamento no início de estágios, garante falhas de projeto antes que se desenvolva. Além disso, é intensa a documentação e planejamento para torná-lo (NATIONAL INSTRUMENTS, 2016).

 O ciclo de vida do Cascata consiste em várias fases que não se sobrepõem, como mostrado na Figura 2.

 O modelo inicia estabelecendo requisitos de sistema e de software e continua com projeto arquitetônico, projeto detalhado, codificação, testes e manutenção. O Modelo Cascata serve como uma base para muitos outros modelos de ciclo de vida. A lista a seguir detalha as etapas para usar o modelo em cascata (NATIONAL INSTRUMENTS, 2016):

Figura 2: Modelo Cascata



Fonte: NATIONAL INSTRUMENTS (2016)

* **Requisitos do sistema**: Estabelece os componentes para a construção do sistema, incluindo os requisitos de hardware, ferramentas de software e outros componentes.
* **Requisitos de software**: Estabelece as expectativas para a funcionalidade do software e identifica quais os requisitos de sistema do software afeta. A análise de requisitos inclui determinar a interação necessária com outras aplicações e bancos de dados, requisitos de desempenho, requisitos de interface do usuário, e assim por diante.
* **Projeto arquitetônico**: Determina a estrutura de um sistema de software para atender às necessidades específicas. Este *design* define os componentes principais e a interação dos componentes, mas não define a estrutura de cada componente. As interfaces externas e ferramentas utilizadas no projeto pode ser determinada pelo *designer*.
* **Projeto detalhado**: Examina os componentes de software definido na fase de concepção arquitetônica e produz uma especificação de como cada componente é implementado.
* **Codificação**: Implementa a especificação do projeto detalhado.
* **Teste**: Determina se o software atende aos requisitos especificados e encontra todos os erros presentes no código.
* **Manutenção**: Aborda os problemas e pedidos de melhorias após os lançamentos de software.

 Em algumas organizações, existe um painel de controle de mudanças mantém a qualidade do produto, revendo cada mudança feita na fase de manutenção. Considera-se a aplicação do modelo cascata completa ao corrigir problemas ou implementar essas solicitações de melhoria (NATIONAL INSTRUMENTS, 2016). Em cada etapa são criados os documentos que explicam os objetivos e descrevem os requisitos de cada fase. No final de cada etapa, realiza-se uma revisão a fim de determinar se o projeto deve prosseguir para a próxima fase (NATIONAL INSTRUMENTS, 2016).

 Muitas pessoas acreditam que este modelo não pode ser aplicado a todas as situações (NATIONAL INSTRUMENTS, 2016). Para indústrias onde as exigências permanecem fixas ao longo do projeto como por exemplo na construção, este modelo é funcional. Porém quando aplicado a uma indústria, como, por exemplo, para o desenvolvimento de software, em que os requisitos podem mudar durante o ciclo de vida do projeto, tem suas desvantagens. Projetos de software estão em constante mudança e nem todos os requisitos podem ser obtidos na frente, as coisas ficam esquecidos, e o negócio está sempre a aprender e descobrir maneiras melhores de fazer as coisas (BLANKENSHIP; BUSSA; MILLETT, 2011). Por exemplo, com o modelo em cascata pura, os requisitos devem ser indicados antes de iniciar o *design* e o projeto completo deve ser declarado antes de iniciar a codificação. Não há sobreposição entre as fases. No desenvolvimento do mundo real, no entanto, pode-se descobrir problemas durante o projeto ou codificação, etapas que apontam erros ou lacunas nos requisitos (NATIONAL INSTRUMENTS, 2016).

 O método cascata não proíbe de voltar a uma fase anterior, por exemplo, retornando a partir da fase de concepção à fase de requisitos. No entanto, isso envolve caro retrabalho (NATIONAL INSTRUMENTS, 2016) e um grande custo para a equipe de desenvolvimento, bem como para o cliente (BLANKENSHIP; BUSSA; MILLETT, 2011). Cada fase concluída requer revisão formal e desenvolvimento de documentação extensa. Assim, descuidos feitos na fase de requisitos são caros para corrigir mais tarde (NATIONAL INSTRUMENTS, 2016). Como o desenvolvimento real vem no final do processo, não se vê resultados para um longo tempo. Este atraso pode ser desconcertante para gestão e clientes. Sendo que a quantidade de documentação é excessiva e inflexível (NATIONAL INSTRUMENTS, 2016). Embora o modelo em cascata tenha suas fraquezas, é instrutivo porque enfatiza fases importantes do desenvolvimento do projeto. Mesmo se não se aplicar este modelo, ele deve considerar cada uma dessas etapas e sua relação com seu próprio projeto (NATIONAL INSTRUMENTS, 2016).

 Segundo Munassar e Govardhan (2010), o modelo cascata possui algumas vantagens e desvantagens, conforme apresenta a Quadro 1.

Quadro 1: Vantagens e desvantagens do Modelo Cascata

|  |  |
| --- | --- |
| VANTAGENS | DESVANTAGENS |
| Fácil compreensão e implementação | Idealiza, não corresponde bem a realidade |
| Amplamente utilizado e conhecido (em teoria) | Não reflete a natureza iterativa exploratória do desenvolvimento |
| Reforça bons hábitos | Irrealista esperar exigências precisas no início do projeto |
| Identifica entregas e marcos | Software é entregue no final do projeto, atrasa descoberta erros de graves |
| Documento conduzido, URD, SRD, etc. publicado padrões de documentação | Difícil de integrar a gestão de risco. |
| Funciona bem com times frágeis | Difícil e caro de fazer alterações em documentos |
|  | Sobrecarga administrativa significativa, caro para pequenas equipes e projetos |
|  | Cascata pura |

Fonte: Do autor

### Modelo de Prototipação

 Um protótipo é uma versão inicial de um sistema de software, usado para demostrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas soluções, ou seja, é uma técnica para auxiliar engenheiros de software e clientes a entender o que está sendo construído quando os requisitos não estão claros (SOMMERVILLE, 2011). É considerada entre os quatro avanços fundamentais potenciais na engenharia de software e é vista como um componente essencial para uso em cada projeto de desenvolvimento para sistemas interativos (BECK, P.; JIANG, J. J.; KLEIN, G., 2006). É fundamental que o desenvolvimento do protótipo seja rápido e iterativo a fim de que os custos sejam controlados e os *stakeholders* do sistema possam experimentá-lo no início do processo de (SOMMERVILLE, 2011).

 De acordo com PRESSMAN (2016), a prototipação pode ser utilizada como um modelo de processo independente, porém é comumente utilizada como uma técnica que pode ser implementada no contexto de qualquer um dos modelos de processo existentes. Independentemente da forma como é aplicado, quando os requisitos não estão claros, o paradigma da prototipação auxilia a compreender melhor o que está para ser construído. A Figura 3 apresenta o ciclo do modelo de prototipagem.

Figura 3: Ciclo do modelo de prototipagem



Fonte: PRESSMAN (2016)

 O ciclo inicia com a comunicação. Realiza-se uma reunião para definir os objetivos gerais do software, identificar requisitos já são conhecidos e esquematizar quais áreas necessitam, obrigatoriamente, de uma definição mais ampla. Uma iteração de prototipação é planejada rapidamente e ocorre a modelagem (na forma de um “projeto rápido”). Um projeto rápido é representação dos aspectos do software que serão visíveis aos usuários finais, levando à construção de um protótipo, que é empregado e avaliado pelos envolvidos, que fornecerão um retorno para aprimorar os requisitos. A iteração ocorre conforme se ajusta o protótipo às necessidades de vários interessados e, ao mesmo tempo, possibilita a melhor compreensão das necessidades que devem ser atendidas (PRESSMAN, 2016).

 Como uma das vantagens desta técnica é que os usuários podem ter uma ideia prévia do sistema final, ao passo que os desenvolvedores passam a desenvolver algo imediatamente. Entretanto, a prototipagem pode ser problemática, pois o protótipo pode não ser necessariamente usado da mesma forma como o sistema final, o testador do protótipo pode não ser um usuário típico do sistema, o tempo de treinamento durante a avaliação do protótipo pode não ser suficiente, pode ser impossível ajustar o protótipo para atender aos requisitos não funcionais, mudanças rápidas durante o desenvolvimento significam que o protótipo não está documentado, padrões de qualidade organizacional geralmente são relaxados para o desenvolvimento do protótipo (PRESSMAN, 2016; SOMMERVILLE, 2011).

 Embora possam ocorrer problemas, a prototipação pode ser efetiva para a engenharia de software. O segredo é definir as regras do jogo logo no início, isso significa que todos os envolvidos devem concordar que o protótipo é construído para servir como um mecanismo para definição de requisitos. Portanto, será descartado (pelo menos em parte) e o software final é arquitetado visando qualidade (PRESSMAN, 2016). As limitações ao método são continuamente estudadas e superadas, incluindo as tecnologias e os problemas de gerenciamento de processos (BECK, P.; JIANG, J. J.; KLEIN, G., 2006).

### Modelo Espiral

 O Modelo Espiral proposto por Barry Boehm, é um modelo de processo de software evolucionário que acopla a natureza iterativa da prototipação com os aspectos sistemáticos e controlados do Modelo Cascata. Este método é uma abordagem realista para o desenvolvimento de sistemas e de software em larga escala, fornecendo potencial para o rápido desenvolvimento de versões cada vez mais completas do software (PRESSMAN, 2016; MUNASSAR; GOVARDHAN, 2010).

 Segundo Boehm (2001), o modelo espiral é um gerador de modelos de processos dirigidos a riscos e é utilizado para guiar sistemas intensivos de software. Consiste em uma abordagem cíclica voltada para ampliar, incrementalmente, o grau de definição e a implementação de um sistema, enquanto diminui o grau de risco do mesmo e ainda consiste em uma série de pontos de controle para assegurar o comprometimento dos indivíduos interessados na busca de soluções de sistema que sejam satisfatórios e praticáveis.

 Dividido em um conjunto de atividades metodológicas definidas pela equipe de engenharia de software, o modelo espiral é representado na Figura 4 (PRESSMAN, 2016).

Figura 4: Modelo Espiral



Fonte: PRESSMAN (2016)

 Assim que o processo inicia, a equipe de software realiza as atividades indicadas por um circuito em torno da espiral no sentido horário, começando pelo seu centro. Os riscos são considerados à medida que cada revolução é realizada. O primeiro circuito em volta da espiral pode resultar no desenvolvimento de uma especificação de produto. As passagens subsequentes em torno da espiral podem ser utilizadas para desenvolver um protótipo e, progressivamente, versões mais sofisticadas do software. Cada passagem pela região de planejamento resulta em ajustes no planejamento do projeto. Após o *feedback* do cliente, ajusta-se os custos, o cronograma e o gerente de projeto ajusta a quantidade de iterações planejadas para completar o software (PRESSMAN, 2016).

 O Modelo Espiral pode ser adaptado para ser utilizado ao longo da vida do processo, desta forma, o primeiro circuito em torno da espiral pode representar um projeto de desenvolvimento de conceitos que inicia no núcleo da espiral e continua por várias iterações, até que o desenvolvimento de conceitos fique completo. Caso o conceito seja desenvolvido para ser um produto final, o processo continua pela espiral pelas bordas e um novo projeto de desenvolvimento de produto começa. Passando por uma série de iterações ao longo da espiral, o novo produto evoluirá. Posteriormente, uma volta em torno da espiral pode ser usada para representar um projeto de aperfeiçoamento do produto. Em sua essência, a espiral, caracterizada dessa maneira, permanece em operação até que o software seja retirado (PRESSMAN, 2016).

 Algumas vantagens e desvantagens deste modelo devem ser levadas em consideração (PRESSMAN, 2016; MUNASSAR; GOVARDHAN, 2010):

 Vantagens:

* Alta quantidade de análise de risco;
* Bom para projetos grandes e de missão crítica;
* O software é produzido no início do ciclo de vida do software;
* O desenvolvedor e o cliente compreender e reagem melhor aos riscos em cada nível evolucionário;
* Torna possível a aplicação da prototipação em qualquer estágio do processo evolutivo do produto;
* Reduz os riscos antes de se tornarem problemáticos.

 Desvantagens:

* Pode ser um modelo caro para usar;
* A análise de riscos exige uma especialização altamente específica;
* O sucesso do projeto é altamente dependente da fase de análise de risco;
* Não funciona bem para pequenos projetos;
* Pode ser difícil convencer os clientes de que a abordagem revolucionária é controlável;
* Ela exige considerável especialização na avaliação de riscos e depende dessa especialização para seu sucesso.
* Se um risco muito importante não for descoberto e administrado, ocorrerão problemas.

### RUP – *Rational Unified Process*

 No livro que deu origem ao processo unificado, Grady Booch, Ivar Jacobson e James Rumbaugh discutiram a necessidade de um processo de software dirigido a casas de uso, centrado na arquitetura, iterativo e incremental. Propuseram a UML como uma notação de modelagem orientada em objetos, independente de processos de desenvolvimento. Além disto, propuseram o RUP, que utiliza a UML como notação de uma série de modelos que compõem os principais resultados das atividades do processo (PRESSMAN, 2016; SOMMERVILLE, 2011; FILHO, 2000).

 RUP é uma tentativa de aproveitar os melhores recursos e características dos modelos tradicionais de processo de software, mas caracterizando-os de modo a implementar muitos dos melhores princípios do desenvolvimento ágil de software. Normalmente é descrito em três perspectivas (PRESSMAN, 2016; SOMMERVILLE, 2011):

* Dinâmica, que mostra as fases do modelo ao longo do tempo;
* Estática, que mostra as atividades realizadas no processo;
* Prática, que sugere boas práticas a serem usadas durante o processo.

 A maioria das descrições do processo unificado combina as perspectivas estática e dinâmica em um único diagrama. É constituído de fases que identifica quatro fases distintas no processo de software, sendo essas fases relacionadas ao negócio, e não a assuntos técnicos (SOMMERVILLE, 2011).

* **Fase de concepção**: justifica a execução de um projeto de desenvolvimento de software, do ponto de vista do negócio do cliente. Realiza a comunicação e o planejamento com o cliente. Identificam-se as necessidades de negócio para o software, propõe-se uma arquitetura rudimentar para o sistema e se desenvolve um planejamento para a natureza iterativa e incremental do projeto decorrente. Deve-se identificar todas as entidades externas que vão interagir com o sistema e definir as interações, utilizando essas informações para avaliar a contribuição do sistema para o negócio. Caso essa contribuição seja pequena, o projeto após essa fase pode ser cancelado (PRESSMAN, 2016; SOMMERVILLE, 2011; FILHO, 2000).
* **Fase de elaboração**: detalhado o produto, o suficiente para permitir um planejamento acurado da fase de construção. Desenvolve-se uma compreensão do problema dominante, se estabelece um *framework* da arquitetura para o sistema, desenvolve-se o plano de projeto e identifica-se os maiores riscos de projeto. Além disso, o plano é revisado cuidadosamente para assegurar que escopo, riscos e datas de entrega permaneçam razoáveis. Normalmente, as modificações no planejamento são feitas nesta oportunidade. Ao término desta fase, deve-se ter um modelo de requisitos para o sistema que possa ser um conjunto de casos de uso da UML, uma descrição da arquitetura ou um plano de desenvolvimento de software (PRESSMAN, 2016; SOMMERVILLE, 2011; FILHO, 2000).
* **Fase de construção**: envolve o projeto, programação e testes do sistema, ou seja, produz-se uma versão completamente operacional do produto. Durante esta fase, as partes do sistema são desenvolvidas em paralelo as integradas. No término desta fase, o sistema de software já deve estar funcionando, bem como a documentação pronta (SOMMERVILLE, 2011; FILHO, 2000).
* **Fase de transição**: última fase do RUP implica na transferência do sistema da comunidade de desenvolvimento para a comunidade de usuários e em seu funcionamento em um ambiente real. Na maioria dos modelos esta fase é ignorada por ser uma atividade cara e às vezes problemáticas. Na conclusão desta fase, deve-se ter um sistema de software documentado e funcionando corretamente em seu ambiente de trabalho operacional (SOMMERVILLE, 2011).

 Segundo PRESSMAN (2016) ainda existe a fase de produção do RUP que monitora o uso contínuo do software, disponibiliza suporte para o ambiente operacional, realiza e avalia relatórios de defeitos e solicitações de mudanças. É provável que, ao mesmo tempo em que as fases de construção, transição e produção estejam sendo conduzidas, já se tenha iniciado o incremento de software seguinte, ou seja, as cinco fases do processo unificado ocorrem de forma concorrente e escalonada. Um fluxo de trabalho de engenharia de software é distribuído ao longo de todas as fases do RUP, ou seja, identifica as tarefas para realizar uma importante ação de engenharia de software e os artefatos produzidos como consequência da finalização de tarefas com êxito. Nem toda tarefa identificada para um fluxo de trabalho do processo unificado é conduzida em todos os projetos de software. A equipe adapta o processo (para ficar de acordo com suas necessidades.

 O RUP não é um processo adequado para todo tipo de desenvolvimento, como por exemplo, desenvolvimento de software embutido. Porém tem-se que as inovações mais importantes do RUP são a separação de fases e *workflows* e o reconhecimento de que a implantação de software em um ambiente do usuário é parte do processo. As fases são dinâmicas e têm metas. Os *workflows* são estáticos e são atividades técnicas podem ser utilizadas durante todo o desenvolvimento para alcançar as metas específicas (SOMMERVILLE, 2011).

### Considerações sobre métodos Tradicionais.

 Os métodos Tradicionais foram extremamente importantes na construção de softwares, pois foram desenvolvidos em uma época que necessitava de uma vasta documentação. Os projetos eram mais complexos devido à falta de capacidade no *hardware* existente e sistemas não intuitivos. Atualmente, estes moldes estão ultrapassados, pois os sistemas são intuitivos, onde elaborar uma documentação, se torna muito caro e sem necessidade.

## Métodos Ágeis

 Em fevereiro de 2001, renomeados programadores se reuniram para criticar os métodos tradicionais de desenvolvimento de software, argumentando que eles eram lentos e não lidavam bem com mudanças de escopo no projeto. No debate sobre como poderiam ganhar agilidade em desenvolvimento de software, que foi construído o Manifesto Ágil, embora o desenvolvimento ágil já estivesse em evidência, foi nessa data que se consolidou um “movimento”. Nesta oportunidade, foram criados os pilares do Manifesto Ágil, sendo eles: indivíduos e interação entre eles mais que processos e ferramentas, software em funcionamento mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos, e responder a mudanças mais que seguir um plano (AGILE MANIFESTO, 2001).

 Métodos ágeis são uma estrutura conceitual para desenvolvimento de software que promove iterações de desenvolvimento ao longo do ciclo de vida do projeto. Atualmente existem diferentes tipos de métodos de desenvolvimento ágil, na maioria das vezes visando minimizar o risco desenvolvendo o software em curtos períodos de tempo (NATARAJAN, SOMASUNDARAM, LAKSHMI, 2013). Assim, se desenvolveram em um esforço para sanar fraquezas reais e visíveis da Engenharia de Software convencional. O desenvolvimento ágil tem diversos benefícios, porém não é indicado a todos os projetos. A metodologia ágil tem como principais referências a metodologia *Scrum,* que está em grande evidência no mercado e a metodologia XP (*extreme* p*rogramming* - programação extrema) (PRESSMAN, 2016).

### *Extreme Programming* (XP)

 *Extreme Programming* foi a abordagem mais amplamente utilizada para desenvolvimento de software ágil. Embora os primeiros trabalhos sobre conceitos e métodos associados a XP ocorreram no final dos anos 1980, o trabalho seminal sobre o tema foi escrito por Kent Back (PRESSMAN, 2016; SOMMERVILLE, 2011).

 XP é uma abordagem para o desenvolvimento, baseada no desenvolvimento e entrega de pequenos incrementos de funcionalidade. Confia na melhoria constante do código, no envolvimento do usuário na equipe de desenvolvimento e na programação sábia do par (MUNASSAR; GOVARDHAN, 2010).

 Segundo PRESSMAN (2016), Kent Beck define um conjunto de cinco valoresque estabelecem as bases para todo trabalho realizado como parte da XP, sendo eles, comunicação, simplicidade, feedback, coragem e respeito. Cada um desses valores é usado como um direcionador das atividades, ações e tarefas específicas da XP.

 *Extreme Programming* envolve uma série de técnicas que refletem os princípios dos métodos ágeis, conforme apresenta a Tabela 2 (SOMMERVILLE, 2011):

* **Planejamento incremental**: Registrar os requisitos em cartões de história e as histórias a serem incluídas em um release são determinadas pelo tempo disponível e sua prioridade relativa. Os desenvolvedores dividem essas histórias em desenvolvimento "tarefas";
* **Pequenos releases**: desenvolver um conjunto mínimo de funcionalidades útil, que fornece o valor do negócio. Releases do sistema são frequentes e gradualmente adicionam funcionalidade ao primeiro realese;
* **Projeto simples**: realizar cada projeto para atender apenas às necessidades atuais;
* **Desenvolvimento test-first**: desenvolver um *framework* de testes iniciais automatizados que será utilizado para escrever os testes de uma nova funcionalidade antes que a funcionalidade em si seja implementada;
* **Refatoração**: refatorar o código continuamente assim que encontrarem melhorias de código, mantendo assim o código simples e manutenível;
* **Programação em pares**: trabalhar em pares, desta forma os desenvolvedores podem verificando o trabalho uns dos outros e prestando apoio;
* **Propriedade coletiva**: trabalhar em todas as áreas do sistema para que não se desenvolvam ilhas de expertise. Todos os conhecimentos e desenvolvedores assumem responsabilidade por todo o código. Qualquer desenvolvedor pode alterar qualquer coisa;
* **Integração contínua**: integrar junto ao sistema como um todo. Após essa integração, todos os testes de unidade do sistema devem passar;
* **Ritmo sustentável**: não são consideradas grandes quantidades de horas-extra, pois o resultado final, muitas vezes, é redução da qualidade do código e da produtividade a médio prazo;
* **Cliente no local**: o cliente deve estar disponível todo o tempo à equipe de XP, ou seja, é um membro da equipe de desenvolvimento e é responsável por levar a ela os requisitos de sistema para implementação.

 O Quadro 2 apresenta as vantagens e desvantagens deste método (MUNASSAR; GOVARDHAN, 2010):

Quadro 2: Vantagens e desvantagens do *Extreme Programming*

|  |  |
| --- | --- |
| VANTAGENS | DESVANTAGENS |
| Para grandes projetos o XP aconselha dividi-los em pequenos projetos, pois sua ideia é trabalhar em pequenos e médios projetos | Difícil escalar grandes projetos onde a documentação é essencial |
| Produz boa coesão da equipe | Programar em pares é caro |
| Enfatiza o produto final | A construção do caso de teste é uma habilidade difícil e especializada |
| Iterativo | Pode ser difícil manter o interesse dos clientes envolvidos no processo |
| Abordagem baseada em testes dos requisitos e garantia da qualidade | Dar prioridade a mudanças pode ser difícil quando há várias partes interessadas |
|  | Manter a simplicidade requer trabalho extra |

Fonte: Do autor

###  Scrum

 O método Scrum foi descrito no ano de 1995 por Ken Schwaber e Jeff Sutherland. Atualmente é a mais utilizada pelas empresas. É uma metodologia simples e de fácil compreensão. O Scrum enfatiza o controle do produto e uma das partes importantes do Scrum é dividir as pessoas em equipes e capacitá-las para realizar as tarefas que estão trabalhando (SVERRISDOTTIRA, INGASONB, JONASSONC, 2014).

 Baseia-se na teoria empírica do controle de processos, ou seja, o conhecimento vem da experiência e toma decisões baseadas no que é conhecido. O Scrum emprega uma abordagem incremental iterativa para otimizar a previsibilidade e controlar o risco. Três pilares sustentam cada implementação do controle empírico do processo: transparência, inspeção e adaptação (SCHWABER; SUTHERLAND, 2016).

* **Transparência**: os aspectos significativos do processo devem ser visíveis para os responsáveis pelo resultado. Exige que esses aspectos sejam definidos por uma norma comum, para que os observadores compartilhem um entendimento comum do que está sendo visto (SCHWABER; SUTHERLAND, 2016);
* **Inspeção**: os usuários devem frequentemente inspecionar os artefatos scrum e progredir em direção a um objetivo Sprint para detectar variações indesejáveis. A inspeção não deve ser tão frequente a ponto de atrapalhar o trabalho e são mais benéficas se forem realizadas por inspetores qualificados no ponto de trabalho (SCHWABER; SUTHERLAND, 2016);
* **Adaptação**: O processo ou material a ser processado deve ser ajustado caso o inspetor determine que um ou mais aspectos do processo se desviam para fora dos limites aceitáveis ou se o produto resultante será inaceitável. Este ajuste deve ser realizado o mais rapidamente para minimizar o desvio adicional (SCHWABER; SUTHERLAND, 2016).

 Os princípios do Scrum são empregados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades: requisitos, análise, projeto, evolução e entrega. Em cada atividade, ocorrem tarefas a realizar dentro de um padrão de processo chamado *sprint.* O trabalho realizado dentro de um sprint (o número de sprints necessários para cada atividade metodológica varia dependendo do tamanho e da complexidade do produto) é adaptado ao problema em questão e definido, e muitas vezes modificado em tempo real, pela equipe Scrum. O fluxo geral do processo Scrum é ilustrado na Figura 5 (PRESSMAN, 2016).

Figura 5: Fluxo do processo Scrum



Fonte: PRESSMAN (2016)

 O Scrum ressalta o uso de um conjunto de padrões de processos de software que provaram ser eficazes para projetos com prazos de entrega apertados, requisitos mutáveis e críticos de negócio. Cada um desses padrões de processos define um conjunto de ações de desenvolvimento (PRESSMAN, 2016):

* **Registro pendente de trabalhos (*Backlog*)**: é criado uma lista dos requisitos ou funcionalidades do projeto que fornecem valor comercial ao cliente em ordem de prioridade. Os itens podem ser adicionados em qualquer momento. O gerente de produto avalia o registro e atualiza as prioridades conforme requisitado;
* **Urgências (corridas de curta distância) *sprints***: são unidades de trabalho solicitadas para atingir um requisito estabelecido no *backlog* e que precisa ser ajustado dentro de um prazo já fechado (tipicamente 30 dias). Alterações (por exemplo, itens do registro de trabalho) não são introduzidas durante execução de urgências (*sprint*). Desta forma, o *sprint* permite que os membros da equipe trabalhem em um ambiente de curto prazo, porém estável.
* **Reuniões Scrum**: são reuniões curtas, tipicamente de 15 minutos, realizadas diariamente pela equipe. Realiza-se três perguntas-chave, respondidas por todos os membros da equipe:
* O que você realizou desde a última reunião de equipe?
* Quais obstáculos está encontrando?
* O que planeja realizar até a próxima reunião da equipe?

 O Scrum Master (líder da equipe)conduz a reunião e avalia as respostas de cada integrante. A reunião auxilia a equipe a revelar problemas potenciais o mais cedo possível. Também leva à “socialização do conhecimento” e, portanto, promove uma estrutura de equipe auto-organizada (PRESSMAN, 2016).

 *Demos* é entrega do incremento de software ao cliente para que a funcionalidade implementada possa ser demonstrada e avaliada pelo cliente. É importante notar que a demo pode não ter toda a funcionalidade planejada, mas sim funções que possam ser entregues no prazo estipulado (PRESSMAN, 2016).

 Os papéis, artefatos, eventos e regras do Scrum são imutáveis e, embora a implementação de apenas partes do mesmo seja possível, o resultado não é Scrum. Desta forma, essa metodologia existe apenas na sua totalidade e funciona bem como um recipiente para outras técnicas, metodologias e práticas (SCHWABER; SUTHERLAND, 2016).

### Considerações sobre métodos ágeis

 Os métodos ágeis estão em enfoque no mercado devido a migração das empresas para este modelo de processo. Como é possível observar, os Métodos Ágeis partem para uma filosofia de trabalho, avaliando mais o processo da empresa, buscando gerenciar pequenas equipes de desenvolvimento, pensando mais no trabalho colaborativo e em grupo, deixando de lado toda a documentação. Nos Métodos ágeis, é documentado apenas o que se faz necessário, sendo este um de seus grandes diferenciais.

# Análise do ciclo de vida

 A análise do ciclo de vida engloba uma série de fases e atividades que independente da metodologia escolhida, ocorrem para a realização do seu maior objetivo: entregar o software funcionando corretamente dentro dos prazos e orçamentos previstos para o seu desenvolvimento. Este capítulo tem como objetivo mapear os principais conceitos de requisitos, análise e *design* do projeto, a implementação, testes, manutenção e para os métodos ágeis, gerência de projeto, a fim de obter um software de sucesso (BEZERRA, 2006).

## Requisitos, Especificações

 Requisito e especificações é uma especificação de uma característica ou propriedade que um sistema deve possuir ou fazer, assim como sua restrição de operação. A análise dos requisitos identifica, quantifica, define, prioriza e classifica os principais problemas que o futuro software deve resolver (LINHA DE CÓDIGO, 2016).

 Existem diversas formas de levantamento de requisitos, cada uma com objetivo de explorar diferentes características do problema. Como vive uma imensa variedade de problemas, cada problema com suas características únicas é justificado existir diversos modelos para especificar cada tipo de problema (PRESSMAN, 2016).

### Entrevistas

 É uma das técnicas tradicionais mais simples de utilizar e produz bons resultados logo na fase inicial de obtenção de dados. O entrevistador deve dar espaço ao entrevistado para que o mesmo consiga esclarecer suas necessidades e ideias para a realização do projeto de software (DEVMEDIA, 2016).

 Para Sommerville (2011), existem basicamente dois tipos de entrevista. As entrevistas abertas, em que o entrevistador não tem uma agenda predefinida. É explorado pela equipe de engenharia de requisitos uma série de questões com as partes envolvidas do sistema, desenvolvendo-se assim, uma melhor compreensão das suas necessidades. O outro tipo de entrevista é a entrevista fechada, em que os *stakeholders* respondem um conjunto predefinido de perguntas.

 Os tipos de entrevistas são separados apenas na literatura, geralmente na prática utiliza-se um misto dos dois tipos, pois entrevistas totalmente abertas geralmente não funcionam bem, sendo necessário fazer algumas perguntas para manter a entrevista centrada no sistema que está sendo desenvolvido (SOMMERVILLE, 2011).

 As entrevistas podem ser flexíveis para obter informações e aproximar o engenheiro de requisito do usuário do sistema, deixando que o usuário se sinta participativo no processo de desenvolvimento. Porém pode demandar muito tempo e a tabulação das informações obtidas costumam ser trabalhosas (MARTINS, 2001).

### Workshop

 É uma técnica de elicitação em grupo usada em uma reunião estruturada. Este grupo deve ser formado pela equipe de analistas e uma seleção dos *stakeholders* que melhor representam a organização e o contexto em que o sistema será usado, obtendo um conjunto de requisitos bem definidos. O *workshop* tem como objetivo acionar o trabalho em equipe, onde tem-se um facilitador neutro onde sua função é conduzir o workshop e promover a discussão entre os vários mediadores. O facilitador intercede as tomadas de decisões em processos bem definidos, com objetivo de obter um processo de negociação. Para que o workshop tenha sucesso alguns aspectos devem ser considerados: a postura do condutor do seminário deve ser de mediador e observador; a convocação deve possuir dia, hora, local, horário de início e de término, assunto a ser discutido e a documentação do seminário (DEVMEDIA, 2016).

 Normalmente a técnica *Brainstorming* é utilizada em workshops. Esta técnica é uma apresentação do problema/necessidade a um grupo específico, requerendo assim as soluções. Após os workshops é produzido documentações que refletem os requisitos e decisões tomadas sobre o sistema a ser desenvolvido (DEVMEDIA, 2016).

### Brainstorming

 É uma técnica que tem o objetivo de gerar ideias. O princípio é reunir um conjunto de especialistas em várias sessões para que cada um possa inspirar o outro na criação de ideias que possam contribuir com a solução de um problema. Ideias oriundas e exploradas nas reuniões não devem ser criticadas ou julgadas. É uma prática que pode ser aplicada no início do desenvolvimento, quando pouco do projeto é conhecido e o projeto carece de ideias inovadoras, incentivando os desenvolvedores a serem criativos com as suas soluções (BATISTA; CARVALHO, 2003).

 Para conduzir uma sessão de *brainstorming* as etapas necessárias são (DEVMEDIA, 2016):

* Os participantes devem ser selecionados em função das contribuições diretas que possam dar durante a sessão. Para garantir uma boa representação é importante a presença de pessoas bem informadas, vindas de diferentes grupos;
* O líder da sessão deve explicar os conceitos básicos de *brainstorming* e as regras a serem seguidas durante a sessão;
* Os participantes podem gerar ideias quantas forem exigidas pelos tópicos que estão sendo o objeto do *brainstorming*. Os participantes são convidados, um por vez, a dar uma única ideia. Se alguém tiver problema, passa a vez e espera a próxima rodada.

 Uma pessoa é designada a registrar todas as ideias, de forma que todos os participantes possam visualizá-las. A análise das ideias é realizada na fase final, onde faz-se uma revisão das mesmas. As considerações do grupo são mantidas e classificadas em ordem de prioridade (DEVMEDIA, 2016).

### Questionário

 O uso de questionário para coletar informações é uma técnica que permite ao analista de sistema estudar as atitudes, crenças, comportamento e características de várias pessoas chave da organização que podem ser afetados pelo sistema. Pode-se dizer que existem dois modos de abordar os usuários, um deles são os questionários fechados, questionários podem ser quantificados. Já os questionários com perguntas abertas são perguntas qualitativas, onde o analista de sistema deve analisar e interpretar as respostas. Esta técnica auxilia quando necessitamos extrair as mesmas informações de um grande número de pessoas (KENDALL; KENDALL, 2011).

### Observação

 Nesta técnica realiza-se uma visita ao local em foco com o objetivo de observá-lo, permitindo coletar informações de acordo com o cotidiano das operações e execução dos processos diários do local. Uma das técnicas de observação utilizadas para elucidação de requisitos, muito empregada na área das ciências sociais, é a etnografia (SOMMERVILLE, 2011).

#### Etnografia

 É uma técnica de observação que pode ser utilizada para compreender a forma de operação e ajudar a extrair requisitos que servirão de apoio para esses processos. É designado um analista para fazer uma imersão no ambiente de trabalho em que o sistema será embarcado. Com a observação do dia a dia é possível que o analista observe necessidades implícitas e esse é o grande valor da etnografia, pois visa eliminar as dificuldades dos usuários ao invés de refletir processos formais definidos pela organização (SOMMERVILLE, 2011).

 Geralmente as pessoas não conseguem detalhar seu trabalho. É comum que elas entendam o seu próprio trabalho, mas não consigam fazer uma relação com outras áreas da organização. Fatores como sociais e organizacionais que afetem o trabalho não são claros para os indivíduos, podendo ficar mais claro para os analistas, pois são indivíduos neutros na organização (SOMMERVILLE, 2011).

 Sommerville (2011) afirma que a etnográfica é eficaz para descobrir dois tipos de requisitos:

1. Requisitos derivados da maneira que as pessoas realmente fazem seu trabalho, deixando de lado a forma que deveria ser feito.
2. Requisitos derivados da cooperação e conhecimento das atividades de outras pessoas.

### JAD

 JAD (*Join Application Design*) é um conceito desenvolvido pela IBM que consiste em workshops e sessões de grupo nos quais *stakeholders* e analistas de requisitos discutem as características desejadas do produto.

 O JAD envolve todos os stakeholders disponíveis e busca investigar através de discussões quais problemas existem e quais as possíveis soluções. Como estão envolvidos todos os stakeholders, as decisões podem ser tomadas de forma mais efetiva e sua resolução se torna mais rápida. A diferença de JAD para *Brainstorming* é que geralmente os principais objetivos já estão definidos, na qual as sessões de JAD são tipicamente estruturadas com etapas, ações e papéis definidos previamente (AURUM, A; WOHLIN, C., 2005).

 Tipicamente uma sessão de JAD duram de 1 a 10 dias e todos os integrantes devem participar em tempo integral, para não haver perda de tempo em recapitulações para os ausentes em sessões anteriores. As sessões devem ser conduzidas em locais afastados das organizações, pois interrupções são altamente prejudiciais e todos os participantes devem ser previamente avisados disto (FILHO, 2000).

 Segundo Filho (2000) as sessões para levantamento de requisitos devem compreender as seguintes etapas:

* Mostrar qual a finalidade da sessão, agenda e qual tempo previsto;
* Definição de alto nível dos requisitos, identificar quais as necessidades do cliente, qual o produto deve atender, objetivo do produto, benefícios esperados, possíveis funções e considerações estratégicas;
* Delimitar escopo do produto;
* Definição dos requisitos e esboço de layout das interfaces de usuário do produto;
* Planejamento do JAD de análise, estimativas, identificação dos participantes e elaboração do cronograma;
* Documentação dos problemas e considerações.

 Ao final da sessão, conforme FILHO (2000) deve ser produzido o seguinte material:

* Todos os materiais produzidos durante as etapas;
* O corpo da especificação de requisitos do software;
* Diagramas e fluxos de casos de uso, no modelo de análise;
* Uma apresentação da parte já produzida das especificações de requisitos do software para os responsáveis pela decisão de continuar o projeto.

 Por ser um processo sofisticado, o JAD exige um intenso comprometimento dos envolvidos, embora dure normalmente um período curto, se comparado com o projeto. Deve-se realçar com o cliente os benefícios que o uso do JAD pode trazer, se aplicado da forma correta, para que desta forma libere o respectivo pessoal de suas funções para fazer o levantamento de requisito (FILHO, 2000).

### Prototipagem

 Este método explora aspectos críticos dos requisitos de um produto, implementando de forma rápida um pequeno subconjunto de funcionalidades do mesmo. Realiza-se uma reunião com os interessados e levanta-se alguns requisitos básicos, sendo indicado para estudar as alternativas de interface do usuário, problemas de comunicação com outros produtos e a viabilidade de atendimento dos requisitos de desempenho (DEVMEDIA, 2016).

 As técnicas utilizadas na elaboração do protótipo são várias: interface de usuário, relatórios textuais, relatórios gráficos, entre outras. Desta forma, reduz-se os riscos na construção do sistema, pois o usuário chave já verificou o que o analista captou nos requisitos do produto. Para ter sucesso na elaboração dos protótipos é necessária a escolha do ambiente de prototipagem, o entendimento dos objetivos do protótipo por parte de todos os interessados no projeto, a focalização em áreas menos compreendidas e a rapidez na construção (DEVMEDIA, 2016).

### Casos de Uso

 A técnica de casos de uso surgiu inicialmente no método *Objectory*, sendo esta uma técnica de descoberta de requisitos. Podemos dizer que um caso de uso identifica os atores envolvidos em uma iteração e dá nome ao tipo de iteração. Essa é complementada por informações adicionais que descrevem a interação com o sistema. Informações adicionais podem ser descritas de forma textual ou em mais modelos gráficos, conforme representa a Figura 6, diagrama de sequência ou de estado de UML (SOMMERVILLE, 2011).

 Inicialmente deve-se definir o conjunto de “atores” envolvidos na história. Atores devem ser diferentes pessoas ou dispositivos que irão utilizar o sistema e representam os papeis que pessoas ou dispositivos desempenham enquanto o sistema opera. Ator é qualquer coisa que se comunique com o sistema ou produto e que é externa ao sistema em si e deve possuir uma ou mais metas ao usar o sistema. Dificilmente todos os atores serão identificados logo na primeira iteração (PRESSMAN, 2016).

 Cada caso de uso deve ser documentado com uma descrição textual, podendo estar ligado a outros modelos UML que desenvolverão o cenário com mais detalhes (SOMMERVILLE, 2011).

 O Autor Sommerville (2011) conclui que,

 “A UML é, de fato, um padrão para modelagem orientada a objetos, e assim, os casos de uso e elicitação baseada em casos de uso são amplamente usados para a elicitação de requisitos. [...]”

Figura 6: Diagrama de caso de uso em UML



Fonte: Pressman (2016)

### Histórias de usuário

 Nos métodos ágeis, pode-se dizer que a forma mais comum de levantamento de requisitos são as histórias de usuários, onde utiliza-se cartões para descrever histórias. É priorizado de acordo com o seu valor para o negócio (FOGGETTI, 2015).

 Conforme Pressman (2016), as histórias de usuários são escritas pelo cliente e colocadas em uma ficha. O próprio cliente deve atribuir um valor à história, baseando-se no valor de negócio global do recurso ou função.

 Para Foggetti (2015) as histórias de usuários têm três aspectos, também conhecidos de “3Cs”:

* O primeiro “C” é o próprio cartão onde as histórias são escritas. Não é necessário conter toda a informação do requisito, apenas um resumo e pode ser escrito notas sobre a prioridade.
* O segundo “C” é o diálogo entre cliente e desenvolvedor.
* O terceiro “C” é a confirmação se o produto foi implementado conforme solicitado.

 O conjunto de cartão forma o artefato com a descrição do requisito, podendo-se dizer que a limitação desse método é a falta de detalhamento no requisito (FOGGETTI, 2015).

### Considerações sobre requisitos

 Tratando-se de requisitos é possível afirmar que muito tempo foi investido, pois é vastamente encontrado na literatura, existindo diversas formas de extrair requisitos. Algumas metodologias, como por exemplo JAD, são extremamente complexas e necessitam de um grau de maturidade muito grande, tanto da equipe quanto da empresa, isso para compreender e aceitar algumas diretrizes da metodologia, portanto poucas empresas utilizam tal método. Outras metodologias são mais simples de praticá-las e tem um retorno satisfatório em um curto prazo de tempo.

## Análise

 O termo análise em engenharia de software significa dividir um sistema em componentes analisando como tais componentes interagem, com objetivo de entender como esse sistema irá funcionar. Essa é a etapa na qual os analistas estudam detalhadamente os requisitos já levantados e a partir desse estudo são construídos modelos que representam o sistema a ser construído (BEZERRA, 2006).

 A análise pode ser dividida em duas análises, a orientada a objetos e a estruturada. Os autores Gane e Sarson (1987) abordam que na análise estruturada os principais diagramas são DFD (diagrama de Fluxo de dados), modelo ER (entidade-relacionamento) e Dicionário de dados. Já a análise orientada a objetos utiliza o UML e é citada por Bezerra (2006) que mostra que existe 13 tipos de diagramas, porém os mais comuns são o diagrama de classes, diagrama de casos de uso e diagrama de interação.

### Análise orientada a objetos

 O início da década de 90 foi o período em que surgiu um novo paradigma na modelagem, a análise orientada a objetos. Os principais colaboradores desse paradigma são Sally Shlaer, Stephen Mellor, Rebecca Wirfs-Brock, James Rumbaugh, Grady Booch e Ivar Jacobson (BEZERRA, 2006).

#### UML

 Booch, Jacobson e Rumbaugh propuseram a UML (*Unified Modeling Language*) com uma notação de uma modelagem orientada a objetos, independente de processos de desenvolvimento (FILHO, 2000).

 A *Unified Modeling Language* (UML) é uma modelagem visual de uso geral que pode ser usada para especificação, visualização, construção e documentação de artefatos de um sistema de software. As especificações da UML não definem um processo padrão, mas visam serem úteis com um processo de desenvolvimento iterativo, destinado a suportar a maioria dos processos de desenvolvimento orientado a objetos (RUMBAUGH; JACOBSON; BOOCH, 2005).

 Não existe uma linha definida entre os vários conceitos e construções em UML, porém é possível separá-los em vários pontos de vista. Uma visão é um subconjunto de construções em modelagem UML, que representa o aspecto de um sistema. A Quadro 3 apresenta o panorama UML e os diagramas que as exibem, bem como os conceitos relevantes para cada visualização, porém não devemos considerar essa tabela um conjunto rígido de regras, apenas um guia podendo misturar os pontos de vista (RUMBAUGH; JACOBSON; BOOCH, 2005).

Quadro 3: Técnicas de UML

|  |  |  |  |
| --- | --- | --- | --- |
| ABRANGÊNCIA  | VISUALIZAÇÕES | DIAGRAMAS | CONCEITOS |
| Dinâmico | Visualização estática | Diagrama de classe | Associação, classe, dependência, generalização, interface, realização. |
| Visualização do *Design* | Estrutura interna | Conector, interface, parte, porta, fornecedor de interface, regras, solicitante de interface. |
| Diagrama de colaboração | Conector, colaborador, uso da colaboração, regras. |
| Diagrama de componente | Componentes, dependência, porta, provedor de interface, realização, requisição de interface, subsistema. |
| Visualização da máquina de status | Diagrama de status da máquina | Complemento de transição, fazer atividade, efeito, evento, região, status, transição, gatilho. |
| Visualização de atividades | Diagrama de atividades | Ação, atividade, controle de fluxo, controle de *node,* fluxo de dados, exceções, expansões de regiões, *fork, join,* objeto do *node,* *pin.* |
| Visualização de interações | Diagrama de sequência | Especificação da ocorrência, execução da especificação, interação, fragmento da interação, interação operando, linha do tempo, mensagem, sinal. |
| Diagrama de comunicação | Colaboração, guardar condição, mensagem, regra, sequência numérica |
| Físico | Visualização da implantação | Diagrama da implantação | Artefato, dependência, manifestação, *node.* |
| Modelo de controle | Visualização do modelo de controle | Diagrama de pacote | Importação, modelo, pacote. |
| Perfil | Diagrama de pacote | Limitação, perfil, estereótipo, valor marcado. |

Fonte: (RUMBAUGH; JACOBSON; BOOCH, 2005).

### Análise Estruturada

 Meados da década de 80, os computadores se tornaram ainda mais avançados e baratos. Foi nessa época que surgiu a necessidade por interfaces mais sofisticadas, por consequência, originou a produção de sistemas de software mais complexos. Nesse período surgiu a análise estruturada, que teve como mentores o Edward Yourdon, Peter Coad, Tom DeMarco, James Martin e Chris Gane (BEZERRA, 2006).

#### Diagrama de Fluxo de Dados

 O Diagrama de Fluxo de Dados (*data flow diagram*, DFD) tem uma visão de entrada, processo e saída de um sistema. Pode-se dizer que embora seja usado pelo UML, DFD não é uma parte formal da UML, apenas um complemento. O diagrama de fluxo é mostrado de forma hierárquica, sendo o primeiro modelo de fluxo de dados que representa o sistema como um todo e os diagramas subsequentes refinam o diagrama de contexto, entregando um detalhamento progressivo em cada nível subsequente (PRESSMAN, 2016).

 O Diagrama de Fluxo de Dados permite que seja desenvolvido modelos do domínio de informações e domínio funcional. À medida que o DFD é refinado com cada vez mais níveis de detalhamento, devemos realizar uma decomposição funcional implícita do sistema. Ao mesmo tempo, o refinamento do DFD nos leva a um correspondente refinamento de dados à medida que se avança nos processos que moldam a aplicação (PRESSMAN, 2016).

#### Dicionário de Dados

 O Dicionário de Dados é um conjunto de dados que aborda sobre dados. A ideia seria oferecer informações sobre a definição, a estrutura e a utilização de cada elemento de dados que o sistema irá utilizar. O elemento de dado é a unidade de dados que não pode ser decomposta. Ao se referir em dicionário de dados é importante saber que existem duas formas hierárquicas de dados: a organização hierárquica tradicional e a emergente (GANE; SARSON, 1987).

#### Diagrama Entidade-Relacionamento (DER)

 A principal notação para modelagem de dado conceitual é o diagrama entidade-relacionamento, com o propósito de representar objetos de dados e suas relações sendo que cada entidade é representada por pelo menos uma tabela de dados e geralmente expressa um depósito de dados do DFD (REZENDE, 2005).

 Originalmente proposto por Peter Chen para o projeto de sistemas de banco de dados relacionados, mas foi estendido para outros autores. O principal objetivo do DER é representar objetos de dados e seus relacionamentos (PRESSMAN, 2016).

### Considerações sobre análise

 A análise de requisitos busca estruturar os requisitos que foram levantados. Embora seja uma área importante para correto andamento do projeto, é possível afirmar que pouco foi estruturado na literatura, sendo grande parte das referências antigas e com pouca riqueza de detalhes. Assim sendo, é possível dizer que não tenha sido investido tantos esforços nesta área, pois se torna muito caro criar a documentação necessária, portanto geralmente é estruturado pela equipe que faz o levantamento de requisitos.

## *Design* de software

 Diversos autores optam por não dividir a análise e *design*, geralmente na literatura é encontrado os dois processos com um só. Nessa fase do projeto, é determina-se como o sistema irá funcionar, visando atender todos os requisitos. Essa etapa irá depender dos recursos tecnológicos existes, então junto da análise são adicionadas as denominadas “restrições da tecnologia”. Alguns aspectos que são considerados nessa fase são a arquitetura do sistema, padrão de interface gráfica, a linguagem de programação, o SGBD, etc. (BEZERRA, 2006).

### Modelo de Entidade Relacional

 Este modelo segue algumas das mais importantes informações de semântica sobre o mundo real. Baseia-se na percepção de um mundo como um conjunto de objetos, que podemos denominar de entidades, e ainda pelo conjunto dos relacionamentos entre essas entidades. A característica das entidades é conhecida como atributos. Esta é uma técnica especial e introduzida para projetos de banco de dados (CHEN, 1976).

### *Model-Driven Architecture* (MDA)

 A arquitetura orientada a modelos (MDA), propõe a criação modelos em diversos níveis de abstração, separando os interesses de implementação de arquitetura que pretendem ser implementados. A MDA utiliza o modelo conceitual de uma aplicação para geração de outro em determinada plataforma, com isso é possível reduzir drasticamente o número de implementações de sistema para o mesmo domínio da aplicação (SILVA; SAMPAIO; PEZZIN, 2006).

### *Design Pattern*

 O *Design Pattern* nomeia, resume e identifica os aspectos-chave de uma estrutura de *design*, o que se torna útil para criar um *design* orientado a objeto reutilizável. O *Design Pattern* identifica as classes, instâncias participantes, suas funções, colaborações e distribuição de responsabilidade. Cada *Design Pattern* deve se focar em um problema na orientação de objeto em particular, deve descrever quando se aplica e ainda se pode ser aplicado em vista das outras limitações de projeto (GAMMA, E. et al., 1997).

### Refatoração

 A fatoração é uma atividade sugerida por diversas metodologias ágeis, sendo uma técnica simples de reorganização que pode simplificar o projeto de um componente sem mudar sua função ou comportamento. Quando refabricamos um software, o projeto é examinado, visando remover termos redundantes, elementos que não estão sendo utilizados, completar algoritmos incompletos ou remoção dos desnecessários, estruturas de dados mal construídas ou inapropriadas, ou qual quer outro ponto que possa ser aprimorado no projeto do software. O resultado dessa implementação será um software mais fácil de se integrar, testar e manter (PRESSMAN, 2016).

### Considerações sobre *Design* de Software

 *Design* de software é aconselhado para empresas que queiram garantir que não haja imprevistos com relação a viabilidade da tecnologia empregada no projeto. Como as tecnologias tanto de *hardware* quanto de software evoluíram muito nos últimos dez anos, fica visível que pouco foi investido em *design* de software, pois as referências são mais antigas e remetem a época que havia grande limitação de *hardware.*

## Construção

 É nessa etapa da produção do software que o sistema é codificado, ou seja, ocorre a tradução da descrição computacional criada na fase do projeto em código executável através de uma ou mais linguagens de programação (BEZERRA, 2006). Na literatura de engenharia de software não se encontra diversos modelos de implementação de software, sendo o “*Pair Programming*” ou programação em par um dos poucos expressos na literatura. Porém existem em artigos específicos da área é possível encontrar modelos de construção.

### Programação em Par

 A programação em par foi pensada no modelo XP, onde é recomendado que dois programadores trabalhem juntos, na mesma estação de trabalho para criar código. Essa técnica fornece um mecanismo de resolução de problema em tempo real, pois duas cabeças trabalham melhor que uma e o código é revisado à medida que é criado. A medida que a dupla completa o trabalho, o código que foi desenvolvido é integrado ao projeto (PRESSMAN, 2016). Existem ainda variações na forma de desenvolvimento em par.

## Testes

 A tarefa de testar software não é algo simples. Pode-se dizer que em algumas situações é extremamente difícil elaborar bons casos de teste, se mostrado mais simples de produzir o software do que testá-lo. Desta forma, muita sistematização e controle são necessários para que a atividade de teste de software deixe de ser uma tarefa difícil e ingênua para se tornar uma atividade de engenharia com resultados efetivos e previsíveis (WAZLAWICK, 2013).

### Teste de Unidade

 O teste de unidade busca focar na validação nas menores unidades do projeto do software. Deve ser usado como guia a descrição do projeto no nível de componente, caminhos de controle importantes também são testados a fim de descobrir erros dos limites do módulo. A complexidade relativa dos testes e os erros que revelam é restrita pelo escopo restrito situado para o teste de unidade. Os testes focam na lógica interna do processo e as estruturas de dados dentro dos limites de um componente, podendo ser conduzido em conjunto para diversos componentes (PRESSMAN, 2016).

#### Teste de Unidade para Software Orientado a Objeto

 Quando criamos softwares orientados a objetos, o conceito de teste unitário se modifica, pois, o encapsulamento controla a definição de classes e objetos, ou seja cada classe (objeto), empacota atributos (dados) e as operações que manipulam esses dados. Uma classe encapsulada é usualmente o foco do teste de unidade. Não podemos testar uma única operação isoladamente, mas como parte de uma classe (PRESSMAN, 2016).

 Segundo Sommerville (2011), quando testamos as classes dos objetos devemos projetar os testes para fornecer uma cobertura de todas as características do objeto, ou seja, devemos testar todas as operações associadas ao objeto, definir e verificar o valor de todos os atributos associados ao objeto e colocar todos os objetos em todos os estados possíveis.

### Teste de Integração

 Após os testes unitários é importante garantir que o software funcione como um todo, pois mesmo tendo certeza que todos os módulos individualmente estão operando, não temos certeza que irão operar juntos, sendo possível que haja perda de dados através de interfaces, um componente pode ter um efeito inesperado ou adverso sobre o outros, etc. O teste de integração é uma técnica sistemática para construção de arquitetura de software, ao mesmo tempo em que realizamos testes para descobrir erros associados às interfaces. Essa técnica sugere a construção de uma estrutura de programa determinada pelo projeto a partir de componentes testados em unidades (PRESSMAN, 2016).

 Existem diversos tipos de teste de integração. A integração big-bang, onde cria-se diferentes classes ou componentes separados e depois integra-se tudo junto, sendo uma técnica não incremental. A integração *bottom-up*, que consiste em integrar módulos de mais baixo nível, ou seja, os módulos independentes, depois integrando os módulos de nível imediatamente mais altos. Existe também a integração *top-down*, onde integra-se inicialmente os módulos de níveis mais altos, deixando os básicos para o fim. A última técnica, a mais complexa, é a técnica *sandwich,* que consiste em utilizar as duas formas anteriores a *top-down* e a *bottom-up*, reduzindo o índice de problema das duas, porém seu planejamento é muito complexo (WAZLAWICK, 2013).

#### Teste de Fumaça

 Segundo Pressman (2016), teste de fumaça é uma abordagem de testes de integração, sendo muito utilizada para projetos de prazo crítico. Foi projetada como um mecanismo de marca-passo, pois permite que a equipe de software possa avaliar o projeto frequentemente, onde sua abordagem abrange algumas atividades:

* Componentes de software que foram codificados são integrados em uma “construção” (*build).*
* Uma bateria de testes é criada para exibir erros de impedem a *build* de executar corretamente sua função.
* A *build* é integrada com outras *builds*, e o produto com as suas funções atuais deve passar diariamente pelo teste de fumaça.

### Teste de Validação

 O teste de validação inicia quando os testes de integração terminam, quando componentes individuais já foram testados e o software já está completamente montado e erros de interface já foram descobertos e corrigidos. O teste é focado na experiência de usuários, onde é validado ações visíveis e saídas do sistema. A definição mais simples é que a validação tem sucesso quando o software funciona de uma maneira que pode ser razoavelmente esperada pelo cliente (PRESSMAN, 2016).

### Teste de Sistema

 Teste de sistema envolve a integração de componentes para criar uma versão operacional do sistema e, em seguida, é executado o teste de sistema integrado. Os testes de sistema têm como finalidade analisar se os componentes são compatíveis, se interagem corretamente e transferem os dados certos no momento certo, por suas interfaces (SOMMERVILLE, 2011).

 É possível programar a execução de testes automatizado, utilizando um módulo de programa que faça chamados diretamente no controlador do sistema e, dessa forma, teste todas as condições de sucesso ou fracasso dos passos do sistema, ou ainda de forma manual, através da interface do sistema para executar operações do sistema (WAZLAWICK, 2013).

#### Teste de Recuperação

 O teste de recuperação é um teste que força o sistema a falhar de várias formas e analisa se a recuperação do sistema é executada corretamente sem perda de dados (PRESSMAN, 2016).

#### Testes de Segurança

 O objetivo é validar se os mecanismos de proteção incorporados no sistema realmente irão protegê-lo contra acessos indevidos (PRESSMAN, 2016).

#### Teste por Esforço

 O teste por esforço utiliza recursos do sistema de maneira anormal, utilizando os recursos em grande quantidade, frequência ou volumes anormais. Basicamente o testador busca quebrar o programa (PRESSMAN, 2016).

#### Teste de Desempenho

 O teste de desempenho é criado para testar o desempenho em tempo de execução do software, no contexto de um sistema integrado. Ele deve ser feito em todas as etapas do processo de teste, porém só teremos o desempenho real quando todos os elementos do sistema estiverem integrados (PRESSMAN,2016).

#### Teste de Disponibilidade

 O teste de disponibilidade, também conhecido por teste de configuração, executa o software em cada ambiente no qual ele deve operar e o teste deve examinar os procedimentos de instalação e software de instalação especializados que serão executados pelos clientes (PRESSMAN, 2016).

### Teste Caixa-Branca

 Teste de caixa-branca também é conhecido como testes estruturais são validações executadas com conhecimento do código implementado, ou seja, validam a estrutura do programa em si. Os testes estruturais não irão avaliar todo o funcionamento do sistema, mas as mais representativas (WAZLAWICK, 2013).

### Teste Caixa-Preta

 Teste de caixa-preta também chamado de teste funcional ou comportamental, são testes focados nos requisitos funcionais do software. As técnicas de teste de caixa-preta admitem derivar séries de condições de entrada que devem utilizar completamente todos os requisitos para um programa. É importante salientar que os testes de caixa-preta não são uma alternativa às técnicas de caixa-branca. Trata-se de uma abordagem complementar, com a possibilidade de descobrir uma classe diferente das obtidas com métodos de caixa-branca (PRESSMAN, 2016).

### Considerações sobre testes

 Existem diversas técnicas de execução de teste, geralmente para a aplicação de técnicas de teste sugere-se que seja executado de forma encadeada, ou seja, uma após a outra, de forma a garantir que o produto final seja entregue sem erros aos usuários. Foi estruturada de maneira brilhante pelo autor Pressman, sendo citado diversas vezes dentro desse assunto.

## Manutenção

 A atividade de manutenção é um conjunto de alterações que o software sofre depois de ter sido finalizado. Esta fase ocorre no momento em que o software está em operação e necessita de manutenção, dessa forma, as atividades de manutenção são consideradas tão relevantes e necessárias quanto as atividades de construção. Atualmente a palavra “manutenção” vem sendo substituída por “evolução”, isso se deve ao fato de que as mudanças do software na fase de produção não buscam mantê-lo como está, mas fazê-lo evoluir de forma a se adaptar aos novos requisitos ou ainda a corrigir possíveis defeitos (WAZLAWICK, 2013).

### Manutenção Corretiva

 Conforme Wazlawick (2013) essa modalidade de manutenção busca corrigir os defeitos do software, podendo ser classificada de duas maneiras: manutenção de erros conhecidos e manutenção para detecção e correção de novos erros. As manutenções de erros conhecidos geralmente são registradas e um documento de considerações operacionais ou em notas de versão, de forma que os usuários possam contornar as falhas e evitar maiores transtornos. Na manutenção para detecção e correção de novos erros, os novos erros podem ser descobertos por usuários ao longo do uso do software, quando relatados devem ser incluídos no relatório de versão do software ou ainda dependendo da política da empresa, deve ser levado ao setor de manutenção para correção imediata.

### Manutenção Adaptativa

 A manutenção adaptativa se torna inevitável quando se trata de softwares, pois os requisitos de clientes e usuários mudam com o passar do tempo. Surgem novos requisitos e tecnologias, modificam-se as leis e normas se modificam, etc. (WAZLAWICK, 2013).

### Manutenção Perfectiva

 É uma manutenção que não visa alterar requisitos, mas sim características de desempenho do sistema, ou seja, aprimora os códigos do programa a fim de deixá-lo mais ágil, limpo, organizado, aplica conceitos mais ágeis de programação e realiza a refatoração de telas a fim de facilitar a experiência do usuário com o software (WAZLAWICK, 2013).

### Manutenção Preventiva

A manutenção preventiva é aplicada para resolver problemas em potencial. Verifica a existência de um problema e o corrige antes que ocorra a reclamação do cliente. Desta forma, evita-se o desagrado com o cliente ou até mesmo problemas futuros. Pode ser realizada através de atividades de reengenharia (WAZLAWICK, 2013).

### Considerações sobre manutenção

 Com relação a manutenção é possível afirmar que poucos autores estruturam ou descrevem técnicas sobre o assunto na literatura. Ao revisar os tipos de manutenção é possível refletir que esta deveria ser uma metodologia a parte do projeto, pois em alguns casos, irá necessitar de levantamento de requisitos, analise, *design*, construção/implementação e testes, e em outros casos, será necessário o envolvimento da equipe de desenvolvimento, não apenas com a equipe de manutenção, se tornando um empecilho que deve ser previsto para atender os prazos estipulados no projeto futuro.

# construção do *FRAMEWORK*

 A ideia da construção do *framework* se deu através da busca na literatura por uma ferramenta que fosse capaz de indicar os métodos que pudessem ser utilizados ao escopo do projeto. Como tal ferramenta não foi encontrada, observou-se que a mesma poderia trazer benefícios para a Engenharia de Software e desta forma, elaborou-se uma ferramenta capaz de auxiliar na tomada de decisão de quais as técnicas mais apropriadas para aplicar dentro de um projeto de desenvolvimento de software, levando em consideração a maturidade da equipe que está desenvolvendo, o tempo disponível do projeto, maturidade da empresa de software e a maturidade da empresa que está contratando.

 A ferramenta foi estruturada utilizando-se no Excel uma série de planilhas onde o usuário responderá perguntas referentes ao projeto e a ferramenta realizará o cálculo, de acordo com a resposta dos usuários, referente a qual o modelo mais indicado para a equipe utilizar. Os modelos sugeridos na planilha estão organizados de acordo com o grau de complexidade e dificuldade para serem implementados.

 A ferramenta é utilizada pelo usuário sem necessidade de consultoria, desta forma, criou-se um protocolo de usuário, onde explica como o *framework* funciona, qual a sua finalidade e como compreender os resultados, sendo um documento crucial para utilização da ferramenta.

## Página principal do *framework*

 A página principal do *framework*, visualizada na Figura 7, apresenta a proposta da ferramenta, seu objetivo, a compreensão dos níveis, as instruções de como iniciar a utilização da ferramenta e explica a finalidade das métricas para produtos novos e métricas para atualização. Botões de navegação foram adicionados para direcionar o usuário ao próximo passo da ferramenta.

 A ferramenta se baseia no nivelamento do escopo do projeto, conforme apresenta o Quadro 4. Desta forma, criou-se 5 níveis para avaliar todos os itens a serem atendidos para a escolha do melhor método, onde os mesmos seriam suficientes para realizar a avaliação. Os níveis variam de 1 (nível mais baixo) ao 5 (nível mais alto), sendo os usuários responsáveis pela classificação.

**Figura 7: Página principal do *framework***



Fonte: Do autor.

**Quadro 4: Compreensão dos níveis**

|  |  |
| --- | --- |
| Compreensões dos níveis | Para o uso do *framework*, é necessária a avaliação crítica quanto aos níveis de sua equipe e empresa. A avaliação é feita por pontos de 1 a 5. |
| 1 | = Nível baixo, nunca, ou de forma alguma; |
| 2 | = Nível básico, ou raramente; |
| 3 | = Nível intermediário, ou não muito; |
| 4 | = Nível avançado, muito, ou frequentemente; |
| 5 | = Nível Expert, extremamente, ou muito frequentemente; |

**Fonte: Do autor.**

## Perguntas ao usuário do *framework*

 **As perguntas aos usuários do *framework* foram elaboradas para criar um parâmetro de análise a fim de garantir que o método sugerido se encaixe no escopo do usuário da ferramenta.**

 **As perguntas foram divididas em duas propostas, métricas para produtos novos e métricas para atualização, para que a ferramenta esteja preparada para se adequar ao projeto do usuário. As perguntas referentes as métricas para produtos novos descritas no Quadro 5 e métricas para atualização descritas no Quadro 6, foram elaboradas com algumas diferenças, pois avaliam situações diferentes. O usuário da ferramenta responderá as perguntas dos quadros conforme a avaliação que o mesmo realiza de seu projeto, utilizando a escala de níveis.**

 Verificou-se ao longo dos estudos realizados que alguns tópicos são mais relevantes que outros, desta forma, as perguntas foram separadas em grupos, sendo **divididos em maturidade da equipe, tempo disponível para realização do projeto, maturidade da empresa que está desenvolvendo o software e maturidade da empresa contratante.**

 Cada pergunta em seu grupo de perguntas possui um nível, classificado pelo usuário e possui um peso diferente, determinado pelo autor do *framework*, para cada uma delas, a fim de determinar seu grau de importância. Cada grupo de perguntas possui uma média, que consiste na soma dos níveis relatados pelos usuários, divido pela quantidade de perguntas do grupo. Essa média também possui um peso, a fim de verificar seu grau de importância entre cada grupo de perguntas.

 **Pode-se tomar como exemplo, para a relevância de alguns tópicos, a importância em avaliar** maturidade da empresa. Avaliando-se a maturidade da empresa, obtemos informações para poder conhecer o cliente, suas necessidades, sua disponibilidade para explicar seus problemas, etc. No entanto, a maturidade da equipe que irá desenvolver o projeto é mais relevante que a maturidade do cliente, pois a equipe precisa ter conhecimento de como extrair os requisitos, fazer a modelagem, desenvolver os testes e aplicar a manutenção.

 **Nas métricas para produtos novos, as perguntas abordadas por cada grupo de perguntas são:**

* **Maturidade da equipe: questiona o nivelamento dos integrantes da equipe, domínio de técnicas para a construção do software, compreensão das metodologias e avaliação da criatividade. Estas perguntas são muito relevantes para que a avaliação realizada pela ferramenta tenha um bom desempenho e para que a ferramenta possa julgar qual o método mais apropriado para a equipe.**
* **Tempo disponível para conclusão do projeto: avalia se toda a equipe está qualificada para o desenvolvimento do projeto, ou seja, se necessita de mais tempo para qualificar a equipe. Questiona se há tempo para que sejam utilizadas novas técnicas de engenharia de software, pois, observou-se ao longo dos estudos, que algumas técnicas levam mais tempo que outras para elaboração. Este item ainda questiona se a primeira versão do software seja livre de erros, necessitando assim de mais tempo para a realização de testes. A complexidade do projeto também é fundamental nesta avaliação, pois quanto mais complexo for o projeto, mais tempo será necessário e a técnica deverá ser mais completa.**
* **Maturidade da empresa que irá produzir o software: as perguntas foram baseadas na avaliação que a MPS.BR -**  Melhoria de Processos do Software Brasileiro realiza. Neste grupo de perguntas, avalia-se se a empresa possui algum processo definido de gestão, a política da empresa, seus recursos, qualificação de seus colaboradores, status das demandas, se há descrição do processo definido e se coleta métricas para melhorias futuras do processo. Estes questionamentos servem para verificar se a empresa está orienta seus colaboradores a buscar novos métodos e se oferece suporte para a mesma.
* Maturidade da empresa que deseja o software: as perguntas deste grupo visam avaliar a empresa que deseja o software, verificando se a mesma já possui um software para sua função ou se deseja um software melhor. Avalia a disponibilidade do cliente para o levantamento de requisitos, compreensão do cliente quanto seu processo interno e se o cliente está aberto a mudanças. São perguntas relevantes pois influenciam no tempo para a execução do projeto e na técnica que a equipe aplicará com o cliente.

 **Nas métricas para atualização, as perguntas abordadas por cada grupo de perguntas são:**

* **Maturidade da equipe: as perguntas para esse grupo não sofrem alteração com relação as métricas para produtos novos, sendo relevantes para ambas as métricas.**
* **Tempo disponível para conclusão do projeto: avalia se toda a equipe está qualificada para o desenvolvimento do projeto, ou seja, se necessita de mais tempo para qualificar a equipe. Questiona se há tempo para que sejam utilizadas novas técnicas de engenharia de software, pois, observou-se ao longo dos estudos, que algumas técnicas levam mais tempo que outras para elaboração. Este item ainda questiona qual o grau de urgência nessa atualização, pois dependendo do grau de urgência é necessário a correção ou alteração o mais rápido possível. A complexidade da atualização também é fundamental nesta avaliação, pois quanto mais complexo for o projeto, mais tempo será necessário e a técnica deverá ser mais completa.**
* **Maturidade da empresa que irá produzir o software: as perguntas para esse grupo não sofrem alteração com relação as métricas para produtos novos, sendo relevantes para ambas as métricas.**
* Maturidade da empresa que deseja o software: as perguntas deste grupo visam avaliar a empresa que deseja o software, verificando se a mesma está habituada com o software. Avalia a disponibilidade do cliente para esclarecer em qual parte do sistema necessita de atualização, o grau de necessidade do cliente para realizar a alteração e se o cliente está aberto a mudanças. São perguntas relevantes pois influenciam no tempo para a execução do projeto e na técnica que a equipe aplicará com o cliente.

**Quadro 5: Perguntas para produtos novos**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Maturidade da Equipe** | **Tempo disponível para conclusão do projeto** | **Maturidade da empresa que irá produzir o software** | **Maturidade da empresa que deseja o software** |
| Qual o nível da equipe que irá levantar os requisitos? | Qual o nível de capacitação da equipe envolvida no projeto de desenvolvimento? | A empresa possui uma política organizacional para planejar e executar o processo?  | A empresa já utiliza algum software específico para sua função? |
| Qual o nível do gerente de projetos? | O tempo para execução do projeto permite que sejam utilizadas novas técnicas com relação à engenharia de software?  | A empresa possui recursos suficientes para executar o processo, desenvolver os produtos de trabalho, e fornecer serviços do processo? | Qual a disponibilidade do cliente para levantamento dos requisitos? |
| Qual o nível dos programadores? | É necessário que a primeira versão do software esteja completa e livre de erros? | A empresa treina pessoas que executam ou suportam o processo quando necessário? | Qual o nível de compreensão do cliente quanto ao seu processo interno? |
| Qual o nível da equipe de teste? | Qual o nível de complexidade do projeto? | A empresa examina as atividades, o *status* e os resultados do processo com o gerenciamento de alto nível para resolver problemas? | O cliente está aberto a mudanças? |
| Qual o nível da equipe de suporte? |  | A empresa mantém uma descrição do processo definido? |  |
| Qual o nível de domínio na linguagem de programação? |  | A empresa coleta métricas e informações de melhorias derivadas do planejamento e execução do projeto para oferecer suporte, melhoria dos recursos e do processo da organização? |  |
| Qual o nível de domínio das técnicas e ferramentas de testes? |  |  |  |
| Qual o nível de domínio das técnicas de levantamento de requisitos e análise dos requisitos? |  |  |  |
| Qual o nível que a equipe possui de experiência nos modelos tradicionais de desenvolvimento de software? |  |  |  |
| Qual o nível que a equipe possui de experiência nos modelos ágeis de desenvolvimento de software?  |  |  |  |
| Qual o nível da equipe na Criatividade em: buscar soluções, resolver problemas, e desenvolver produtos novos? |  |  |  |

Fonte: Do autor

**Quadro 6: Perguntas para atualização**.

|  |  |  |  |
| --- | --- | --- | --- |
| **Maturidade da Equipe** | **Tempo disponível para conclusão do projeto** | **Maturidade da empresa que irá produzir o software** | **Empresa que utiliza o software** |
| Qual o nível da equipe que irá levantar os requisitos? | Qual o nível de capacitação da equipe envolvida no projeto de desenvolvimento? | A empresa possui uma política organizacional para planejar e executar o processo?  | A empresa já está adaptada com o software? |
| Qual o nível do gerente de projetos? | O tempo para execução do projeto permite que sejam utilizadas novas técnicas com relação à engenharia de software? | A empresa possui recursos suficientes para executar o processo, desenvolver os produtos de trabalho, e fornecer serviços do processo? | Qual a disponibilidade do cliente para esclarecer e mostrar em qual parte do sistema necessita de uma atualização? |
| Qual o nível dos programadores? | Qual o Grau de Urgência nessa atualização? | A empresa treina pessoas que executam ou suportam o processo quando necessário? | Qual o grau de necessidade do cliente para realizar a alteração? |
| Qual o nível da equipe de teste? | Qual o nível de complexidade da atualização? | A empresa examina as atividades, o *status* e os resultados do processo com o gerenciamento de alto nível para resolver problemas?  | O cliente está aberto a mudanças? |
| Qual o nível da equipe de suporte? |  | A empresa mantém uma descrição do processo definido? |  |
| Qual o nível de domínio na linguagem de programação? |  | A empresa coleta métricas e informações de melhorias derivadas do planejamento e execução do projeto para oferecer suporte, melhoria dos recursos e do processo da organização? |  |
| Qual o nível de domínio das técnicas e ferramentas de testes? |  |  |  |
| Qual o nível de domínio das técnicas de levantamento de requisitos e análise dos requisitos? |  |  |  |
| Qual o nível que a equipe possui de experiência nos modelos tradicionais de desenvolvimento de software? |  |  |  |
| Qual o nível que a equipe possui de experiência nos modelos ágeis de desenvolvimento de software?  |  |  |  |
| Qual o nível da equipe na Criatividade em: buscar soluções, resolver problemas, e desenvolver produtos novos? |  |  |  |

**Fonte: Do autor**

## ****Tabela de resultados****

 Após o preenchimento do nível das perguntas, o usuário seguirá para as tabelas de resultados. A tabela de resultados para ambas as métricas apresentam ao usuário qual o melhor método a ser empregado no projeto, através da média ponderada. O modelo da tabela de resultados apresentados no *framework* pela Figura 8, estão representados por um ciclo de vida da construção de um projeto de software. Tal ciclo de vida foi proposto por um dos primeiros modelos, o Modelo Cascata, dividido em requisitos, análise, *design*, construção, testes e manutenção. No entanto, foi necessária uma adaptação do ciclo de vida tradicional.

 Para ser formulado o *framework,* agrupou-se a análise e o *design* em modelagem, pois alguns modelos de *design* são voltados para a construção de estrutura de banco de dados, e, outros, programação orientado a objetos. Agrupando-se a análise e o *design* em modelagem, foi possível determinar o grau de complexidade, deixando uma visão mais ampla. Ainda, a construção foi removida da estrutura do *framework*, pois, encontrou-se apenas um método de implementação de software na literatura, se tornando irrelevante para a proposta da ferramenta.

 A partir desse problema, surgiu a ideia de utilizar a média ponderada que em determinadas situações, os valores têm graus de importância diferentes. A média ponderada é uma média dos valores afetados de pesos diferentes.

 As técnicas foram classificadas na tabela de resultados por grau de complexidade, ou seja, quanto menor o valor da média ponderada, menos a técnica empregada exige da maturidade da equipe, do tempo disponível para conclusão do projeto, da maturidade da empresa que irá produzir o software e da maturidade da empresa que deseja o software.

 As técnicas estão organizadas por grau de dificuldade de aplicação. Esta ordenação foi estruturada através da leitura e análise das técnicas existentes na literatura, feitas durante a elaboração deste trabalho, pois nenhum autor havia classificado pela dificuldade dos modelos.

 Os modelos foram classificados do menor ao maior grau de dificuldade e separados da seguinte forma:

* Requisitos: Entrevista, w*orkshop, Brainstorming,* Casos de uso, História deusuário, Observação, Questionário, Prototipagem e JAD.
* Modelagem; Modelo de entidade relacional, Refatoração, *Design patterns*, MDA.
* Testes: Teste de Unidade, Teste de integração, Teste de validação, Teste de sistema, Teste Caixa-Branca e Teste Caixa-Preta.
* Manutenção: Corretiva, Adaptiva, Preventiva e Perfectiva.

 Para facilitar a visualização do usuário, utilizou-se as cores da sinaleira (verde, amarelo e vermelho) na média ponderada e nas técnicas, onde cada valor da média ponderada obtido representa uma cor e a técnica a ser empregada. O único ciclo que difere é o teste, pois aconselha-se que os testes sejam feitos até o número indicado pela média ponderada, ou seja, deve-se realizar todos os modelos até o valor indicado pela média ponderada.

 Ao longo da elaboração do *framework* verificou-se que algumas perguntas não possuem o mesmo peso que outras. Por exemplo, dentro do ciclo de vida da construção de software, a pergunta “nível da equipe que irá levantar os requisitos” é fundamental para o levantamento de requisitos, porém, não expressa um valor tão significativo para a manutenção, pois, a equipe que realizará o levantamento de requisitos, não será a mesma que realizará a manutenção. Desta forma, cada pergunta possui um peso específico para cada ciclo de vida na construção de software.

 A média ponderada foi realizada através das respostas dos níveis em cada pergunta, realizados pelos usuários, para apresentar os resultados das perguntas, ou seja, o valor da média ponderada determinará qual método deve ser aplicado. Determinam-se pesos diferentes para cada pergunta, dependentes do grau de relevância das mesmas para cada métrica.

 A média ponderada foi calculada através da Equação 1:

$\overbar{x}\_{p}=\frac{\sum\_{i=1}^{n}x\_{i}.p\_{i}}{\sum\_{i=1}^{n}p\_{i}}$ (1)

Onde, $\overbar{x}\_{p}$= média ponderada

$x\_{i}$= valor do nível determinado pelos usuários, de 1 a 5.

$p\_{i}$= peso de cada pergunta

 Ou seja, para obter a média ponderada, realiza-se o somatório do valor de cada nível multiplicado pelo peso de cada pergunta, dividido pelo somatório de todos os pesos.

 Após os resultados, o usuário fará a análise do modelo indicado pelo *framework*, através dos comentários inseridos em cada técnica na tabela de resultados, conforme apresenta a Figura 8. Após a compreensão das técnicas que lhe foram sugeridas pelo *framework*, o usuário poderá empregá-las em seu projeto.

**Figura 8: Planilha de resultados no *framework***



Fonte: Do autor.

### Pesos das perguntas do *framework*

 Com a finalização visual da planilha de resultados, verificou-se que as perguntas não possuem a mesma importância para determinado ciclo de vida da construção do software. Desta forma, realizou-se a média ponderada onde seria possível elencar um valor (peso) para cada pergunta dentro de cada grupo.

 Os pesos das perguntas foram determinados através de análise de cada técnica já citada neste trabalho, determinados valores de 0 (irrelevante) a 10 (muito relevante) para cada pergunta. A pergunta “nível da equipe que irá levantar os requisitos” é relevante para requisitos, porém não esboça valor para a manutenção, por exemplo.

 Criou-se planilhas ocultas e bloqueadas para cada ciclo de vida dentro de cada métrica, com os respectivos pesos de cada pergunta, para não haver alterações indesejadas. A Figura 9 apresenta um exemplo de como foi inserido os pesos para cada pergunta. Estas planilhas tem o objetivo de inserir o peso de cada pergunta e ainda o peso da média de cada grupo de pergunta, para facilitar a manutenção da planilha e a visualização dos pesos de cada grupo de perguntas. Para cada grupo de perguntas, realizou-se uma média ao final de cada grupo, que determina a relevância do das perguntas para cada ciclo. Na Figura 9, em pesos de requisitos para produtos novos, verificamos através do resultado da média em cada grupo de perguntas que a importância das mesmas em maturidade da equipe é mais relevante que nos outros grupos de perguntas. Ainda na Figura 9, criou-se uma pequena tabela com a soma dos pesos para cada grupo de perguntas a fim de facilitar o cálculo da média ponderada, de forma que, ao realizar uma alteração do peso, o resultado da média ponderada se dá automaticamente.

 Para facilitar a compreensão da classificação dos pesos para cada pergunta, criou-se para métricas para produtos novos o Quadro 7, que retrata o peso das perguntas para cada ciclo de vida (requisitos, modelagem, testes e manutenção), com suas respectivas observações quanto a classificação dos pesos. Com o mesmo objetivo, criou-se o Quadro 8 referente aos pesos em métricas para atualização.

**Figura 9: Pesos de requisitos para produtos novos**



Fonte: Do Autor.

**Quadro 7: Pesos para produtos novos**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| **Maturidade da equipe** | Qual o nível da equipe que irá levantar os requisitos? | 10 | 10 | 0 | 0 | É extremamente relevante para requisitos e modelagem, pois avalia o nível de quem realiza o levantamento de requisitos e faz a análise. Não expressa valor para teste de sistema, e manutenção, pois o analista de negócios não participa dessas etapas. |
| Qual o nível do gerente de projetos? | 3 | 3 | 0 | 0 | Tem baixa importância para requisitos e modelagem, pois o gerente de projetos apoia o andamento de ambas. Não expressa valor para teste e manutenção, pois o gerente de projetos não auxilia nessas etapas. |
| Qual o nível dos programadores? | 1 | 0 | 0 | 0 | O programador não atua em praticamente nenhum dos ciclos expostos na ferramenta, porém, necessita compreender os requisitos levantados para conseguir realizar sua função posteriormente.  |
| Qual o nível da equipe de teste? | 0 | 0 | 10 | 0 | É relevante apenas para testes, pois avalia o nível da equipe de teste. A equipe não participará dos demais ciclos.  |
| Qual o nível da equipe de suporte? | 0 | 0 | 0 | 10 | É relevante apenas para manutenção, pois avalia o nível da equipe de suporte. A equipe não participará dos demais ciclos.  |
| Qual o nível de domínio na linguagem de programação? | 0 | 0 | 10 | 10 | É extremamente relevante para teste, pois alguns testes automatizados são construídos através de linhas de código. Quanto a manutenção, fará o suporte e ajustes quando necessário. Ter o domínio da linguagem de programação não interfere nos demais ciclos. |
| Qual o nível de domínio das técnicas e ferramentas de testes? | 0 | 0 | 10 | 0 | É relevante apenas para testes, pois avalia o conhecimento e domínio das técnicas de teste. Ter o domínio das técnicas e ferramentas de testes não interfere nos demais ciclos. |
| Qual o nível de domínio das técnicas de levantamento de requisitos e análise dos requisitos? | 10 | 10 | 0 | 0 | É extremamente relevante para requisitos e modelagem, pois avalia o conhecimento e domínio de técnicas de levantamento de requisitos e análise. Ter o domínio das técnicas de levantamento de requisitos e de sua análise, não interfere nos demais ciclos.  |
| Qual o nível que a equipe possui de experiência nos modelos tradicionais de desenvolvimento de software? | 3 | 5 | 1 | 1 | É relevante em todo o ciclo de vida pois é necessário conhecer a metodologia para ser bem aplicada posteriormente. |
| Qual o nível que a equipe possui de experiência nos modelos ágeis de desenvolvimento de software?  | 3 | 5 | 1 | 1 | É relevante em todo o ciclo de vida pois é necessário conhecer a metodologia para ser bem aplicada posteriormente. |
| Qual o nível da equipe na Criatividade em: buscar soluções, resolver problemas, e desenvolver produtos novos? | 6 | 8 | 10 | 10 | Tem alto nível de importância em todos os ciclos, pois muitas vezes é necessário utilizar a criatividade para desenvolver um projeto mais complexo.  |
| **Tempo disponível para conclusão do projeto** | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| Qual o nível de capacitação da equipe envolvida no projeto de desenvolvimento? | 10 | 9 | 10 | 4 | Tem alto nível de importância, pois uma equipe qualificada não demanda tanto tempo para executar o projeto. Para manutenção não expressam um valor tão alto, pois a pergunta retarda o desenvolvimento do projeto. |
| O tempo para execução do projeto permite que sejam utilizadas novas técnicas com relação à engenharia de software? | 7 | 7 | 10 | 1 | É extremamente relevante para teste, pois envolve muito tempo para desenvolver um teste que utilize todas as técnicas. Para requisitos e modelagem é importante, pois algumas técnicas mais complexas demandam mais tempo para serem executadas. Não é tão relevante para manutenção, a manutenção será realizada apenas se houver um problema posterior. |
| É necessário que a primeira versão do software esteja completa e livre de erros? | 2 | 10 | 10 | 0 | É extremamente importante para modelagem e teste, pois é necessário que a primeira versão esteja bem estruturada e testada. Para requisitos não é tão expressiva, pois se os requisitos forem levantados corretamente, não sofrem alteração. Não é relevante para manutenção, pois ela se dá após a conclusão e utilização do software. |
| Qual o nível de complexidade do projeto? | 5 | 9 | 7 | 1 | É extremamente importante para modelagem, pois é necessário que esteja bem moldado para facilitar a construção do software. Já para requisitos e teste é importante, pois dependendo do escopo do projeto é necessário que os requisitos sejam levantados com cautela e feitos diversos testes para avaliar se está funcionando corretamente. Para manutenção, não tem tanta relevância, pois o suporte não fará a construção do software. |
| **Maturidade da empresa que irá produzir o software** | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| A empresa possui uma política organizacional para planejar e executar o processo?  | 2 | 3 | 1 | 1 | Não expressa um peso alto, pois o *framework* tem a proposta de auxiliar a política organizacional da empresa. |
| A empresa possui recursos suficientes para executar o processo, desenvolver os produtos de trabalho, e fornecer serviços do processo? | 0 | 3 | 1 | 3 | Não expressa um peso alto, pois o *framework* tem a proposta de auxiliar a política organizacional da empresa. |
| A empresa treina pessoas que executam ou suportam o processo quando necessário? | 0 | 5 | 1 | 3 | É relevante para modelagem, pois é uma área complexa. Para as demais, não expressa um peso alto, pois o *framework* tem a proposta de auxiliar na política organizacional da empresa. |
| A empresa examina as atividades, o status e os resultados do processo com o gerenciamento de alto nível para resolver problemas? | 0 | 5 | 1 | 2 | É relevante pois algumas técnicas são mais simples de serem rastreadas e de analisar o status. |
| A empresa mantém uma descrição do processo definido? | 0 | 3 | 1 | 1 | Não expressa um peso alto, pois o *framework* tem a proposta de auxiliar a política organizacional da empresa. |
| A empresa coleta métricas e informações de melhorias derivadas do planejamento e execução do projeto para oferecer suporte, melhoria dos recursos e do processo da organização? | 5 | 3 | 1 | 5 | É relevante pois faz a análise de qual modelo se adequou melhor a determinada equipe. |
| **Maturidade da empresa que deseja o software** | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| A empresa já utiliza algum software específico para sua função? | 8 | 0 | 0 | 0 | É importante apenas para requisitos, pois é nesse momento que existe uma interação com o cliente. Caso a empresa já possua um software específico, facilita o levantamento de requisitos. |
| Qual a disponibilidade do cliente para levantamento dos requisitos? | 10 | 0 | 0 | 0 | É importante apenas para requisitos, pois é nesse momento que existe uma interação com o cliente. Caso o cliente estando disponível facilitará o levantando dos requisitos para a equipe. |
| Qual o nível de compreensão do cliente quanto ao seu processo interno? | 10 | 0 | 0 | 0 | É importante apenas para requisitos, pois é nesse momento que existe uma interação com o cliente. Caso o cliente compreenda seu processo interno, pode citar precisamente sua necessidade. |
| O cliente está aberto a mudanças? | 1 | 0 | 0 | 0 | Não é relevante para os ciclos. Em requisitos tem baixa relevância, porém é necessário neste momento que o cliente solicite que não haja mudanças no que já estava utilizando. |

Fonte: do autor.

**Quadro 8: Perguntas para produtos novos**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| **Maturidade da Equipe** | Qual o nível da equipe que irá levantar os requisitos? | 10 | 10 | 1 | 0 | É extremamente relevante para requisitos e modelagem, pois avalia o nível de quem realiza o levantamento de requisitos e faz a análise. Não expressa valor para teste e manutenção, pois o analista de negócios não participa dessas etapas. |
| Qual o nível do gerente de projetos? | 3 | 3 | 0 | 0 | Tem baixa importância para requisitos e modelagem, pois o gerente de projetos apoia o andamento de ambas. Não expressa valor para teste e manutenção, pois o gerente de projetos não auxilia nessas etapas. |
| Qual o nível dos programadores? | 1 | 0 | 3 | 0 | O programador não atua em praticamente nenhum dos ciclos expostos na ferramenta, porém, necessita compreender os requisitos levantados para conseguir realizar sua função posteriormente.  |
| Qual o nível da equipe de teste? | 0 | 0 | 10 | 0 | É relevante apenas para testes, pois avalia o nível da equipe de teste. A equipe não participará dos demais ciclos.  |
| Qual o nível da equipe de suporte? | 0 | 0 | 0 | 10 | É relevante apenas para manutenção, pois avalia o nível da equipe de suporte. A equipe não participará dos demais ciclos.  |
| Qual o nível de domínio na linguagem de programação? | 0 | 0 | 10 | 10 | É extremamente relevante para teste, pois alguns testes automatizados são construídos através de linhas de código. Quanto à manutenção, fará o suporte e ajustes quando necessário. Ter o domínio da linguagem de programação não interfere nos demais ciclos. |
| Qual o nível de domínio das técnicas e ferramentas de testes? | 0 | 0 | 10 | 0 | É relevante apenas para testes, pois avalia o conhecimento e domínio das técnicas de teste. Ter o domínio das técnicas e ferramentas de testes não interfere nos demais ciclos. |
| Qual o nível de domínio das técnicas de levantamento de requisitos e análise dos requisitos? | 10 | 10 | 0 | 0 | É extremamente relevante para requisitos e modelagem, pois avalia o conhecimento e domínio de técnicas de levantamento de requisitos e análise. Ter o domínio das técnicas de levantamento de requisitos e de sua análise, não interfere nos demais ciclos.  |
| Qual o nível que a equipe possui de experiência nos modelos tradicionais de desenvolvimento de software? | 3 | 5 | 0 | 1 | É relevante em todo o ciclo de vida pois é necessário conhecer a metodologia para ser bem aplicada posteriormente. |
| Qual o nível que a equipe possui de experiência nos modelos ágeis de desenvolvimento de software?  | 3 | 5 | 0 | 1 | É relevante em todo o ciclo de vida pois é necessário conhecer a metodologia para ser bem aplicada posteriormente. |
| Qual o nível da equipe na Criatividade em: buscar soluções, resolver problemas, e desenvolver produtos novos? | 6 | 8 | 8 | 10 | Tem alto nível de importância em todos os ciclos, pois muitas vezes é necessário utilizar a criatividade para desenvolver um projeto mais complexo.  |
|  | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| **Tempo disponível para conclusão do projeto** | Qual o nível de capacitação da equipe envolvida no projeto de desenvolvimento? | 10 | 7 | 10 | 10 | Tem alto nível de importância, pois uma equipe qualificada não demanda tanto tempo para executar o projeto. Para manutenção não expressam um valor tão alto, pois a pergunta retrata o desenvolvimento do projeto. |
| O tempo para execução do projeto permite que sejam utilizadas novas técnicas com relação à engenharia de software? | 7 | 7 | 5 | 1 | É relevante para teste, pois envolve muito tempo para desenvolver um teste que utilize todas as técnicas. Para requisitos e modelagem é importante, pois algumas técnicas mais complexas demandam mais tempo para serem executadas. Não é tão relevante para manutenção, pois a mesma será realizada apenas se houver um problema posterior. |
| Qual o Grau de Urgência nessa atualização? | 5 | 3 | 5 | 4 | É importante para requisitos, pois pode ser uma atualização crucial para o sistema. Portanto, deve ser bem especificada pelo levantamento de requisitos. Para modelagem não é tão relevante, pois o sistema já está estruturado. Para testes e manutenção é relevante pois o sistema deve ser testado para não haver problemas com a integração do sistema e a manutenção, geralmente é quem recebe a demanda de atualização. |
| Qual o nível de complexidade da atualização? | 5 | 9 | 7 | 8 | É extremamente importante para modelagem, pois é necessário que esteja bem moldado para facilitar a construção do software. Já para requisitos e teste é importante, pois dependendo do escopo do projeto é necessário que os requisitos sejam levantados com cautela e realizados diversos testes para avaliar se está funcionando corretamente. Para manutenção é relevante, pois o suporte pode auxiliar na demanda. |
|  | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| **Maturidade da empresa que irá produzir o software** | A empresa possui uma política organizacional para planejar e executar o processo?  | 2 | 4 | 3 | 1 | Não expressa um peso alto, pois o *framework* tem a proposta de auxiliar a política organizacional da empresa. |
| A empresa possui recursos suficientes para executar o processo, desenvolver os produtos de trabalho, e fornecer serviços do processo? | 0 | 1 | 3 | 3 | Não expressa um peso alto, pois o *framework* tem a proposta de auxiliar a política organizacional da empresa. |
| A empresa mantém uma descrição do processo definido? | 0 | 5 | 5 | 3 | É relevante para modelagem, pois é uma área complexa. Para as demais, não expressa um peso alto, pois o *framework* tem a proposta de auxiliar na política organizacional da empresa. |
| A empresa examina as atividades, o *status* e os resultados do processo com o gerenciamento de alto nível para resolver problemas?  | 0 | 7 | 5 | 2 | É relevante pois algumas técnicas são mais simples de serem rastreadas e de analisar o status. |
| A empresa mantém uma descrição do processo definido? | 0 | 3 | 3 | 1 | Não expressa um peso alto, pois o *framework* tem a proposta de auxiliar a política organizacional da empresa. |
| A empresa coleta métricas e informações de melhorias derivadas do planejamento e execução do projeto para oferecer suporte, melhoria dos recursos e do processo da organização? | 5 | 5 | 3 | 5 | É relevante pois faz a análise de qual modelo se adequou melhor a determinada equipe. |
|  | **Perguntas** | **Requisitos** | **Modelagem** | **Testes** | **Manutenção** | **Observações** |
| **Maturidade da empresa que deseja o software.** | A empresa já está adaptada com o software? | 8 | 0 | 0 | 10 | É importante para requisitos e manutenção, pois é nesse momento que existe uma interação com o cliente e o suporte auxilia o usuário com as dificuldades encontradas. Caso a empresa já possua um software específico, facilita o levantamento de requisitos. |
| Qual a disponibilidade do cliente para esclarecer e mostrar em qual parte do sistema necessita de uma atualização? | 10 | 0 | 0 | 5 | É extremamente importante para requisitos, pois é nesse momento que existe uma interação com o cliente. Caso o cliente esteja disponível, facilitará o levantamento dos requisitos para a equipe. Já para manutenção, é importante, pois geralmente é quem recebe a demanda de atualização. |
| Qual o grau de necessidade do cliente para realizar a alteração? | 10 | 0 | 0 | 0 | É importante apenas para requisitos, pois é nesse momento que existe uma interação com o cliente. Caso o cliente compreenda seu processo interno, pode citar precisamente sua necessidade. |
| O cliente está aberto a mudanças? | 5 | 8 | 0 | 0 | Não é relevante para os ciclos. Em requisitos tem baixa relevância, porém é necessário neste momento que o cliente solicite que não haja mudanças no que já estava utilizando. |

Fonte: do autor.

## Protocolo do usuário

 Ao longo da construção do *framework,* percebeu-se que a ferramenta estava muito complexa para que o usuário pudesse utilizá-la sem o auxílio de algum documento. Por isso, elaborou-se um documento que explicasse como a ferramenta deveria ser utilizada e seu funcionamento. A partir dessa necessidade, criou-se o protocolo do usuário (Anexo A), um documento que tem o objetivo de tornar a ferramenta independe, que não necessite que uma explicação presencial do seu funcionamento. Este documento apresenta as explicações referentes a utilização do *framework*, da seguinte forma:

* Objetivo da ferramenta;
* Motivo que foi elaborado o protocolo;
* Qual contexto o usuário deve utilizar;
* Motivo de responder as perguntas;
* Explicação dos resultados;
* Solicitar ao usuário que responda o questionário após a utilização.

## Considerações sobre o *framework*

 A construção do *framework* foi iniciada através das análises dos modelos que seriam inseridos dentro da ferramenta. Posteriormente, foi construído uma ferramenta utilizando a análise realizada previamente. Testes foram realizados para avaliar seu funcionamento. A validação da ferramenta foi realizada através de um estudo piloto com os alunos da disciplina de Engenharia de Software. Desta forma, é possível afirmar que a ferramenta cumpriu com os objetivos específicos descritos no anteprojeto do trabalho de conclusão, que são:

* Mapear práticas tradicionais e ágeis para as fases de ciclo de vida do software;
* Analisar quais práticas podem ser combinadas para auxiliar na eficiência de uma equipe de desenvolvimento de software;
* Organizar um *framework* para composição de processos híbridos;
* Propor recomendações a respeito de quais as práticas mais adequadas em determinados contextos.

 Disponibilizou-se o *framework* através do link: <https://drive.google.com/file/d/0B6Zs1Eir13dDbG1YZGtFejhoQ3c/view?usp=sharing>

# avaliação do *Framework*

## Estudo piloto

 Após a construção do *framework* seria necessário realizar a validação da ferramenta, para analisar se os objetivos deste trabalho seriam alçados. Desta forma, desenvolveu-se um estudo piloto, realizado em uma turma de Engenharia de Software II na Universidade Feevale, com auxílio e orientação da professora Adriana Neves dos Reis. O estudo foi dividido em dois momentos, o primeiro seria individual, onde o aluno poderia explorar a ferramenta. A segunda parte seria em grupo, onde os alunos teriam o mesmo escopo para avaliar e teriam papéis pré-definidos para fazer a avaliação em cima do conhecimento de cada um.

 O estudo piloto foi realizado para analisar se a ferramenta traz algum ganho para Engenharia de Software, e, se é possível diversificar modelos utilizados em metodologias Tradicionais com modelos utilizados em metodologias Ágeis. Para fazer essa avaliação, elaborou-se um questionário onde os alunos responderiam perguntas para análise posterior se os objetivos foram alcançados.

### Análise do ambiente

 Para realizar o estudo, a professora Adriana cedeu parte de uma aula da disciplina de Engenharia de Software II, onde 16 alunos, propuseram-se a participar do experimento. O estudo em um ambiente acadêmico pôde trazer diversas vantagens, pois existem pessoas com as mais variadas características, sendo alguns mais experientes e outros com pouco ou nada de experiência em desenvolvimento e manutenção de software*.*

 Anteriormente a utilização da ferramenta, os alunos trabalharam em sala de aula um estudo de caso, que seria útil posteriormente para a utilização do *framework*.

 Individualmente, os alunos utilizaram a ferramenta a fim de se adaptar e conhecê-la para poder julgá-la na avaliação realizada. Em grupo, um dos integrantes preencheu a planilha e os demais o auxiliaram. Para realizar o estudo, foi solicitado a formação de três grupos, onde um grupo trabalharia com metodologia Ágil, outro grupo com metodologia Tradicional e o outro grupo livre, porém devido à falta de tempo para avaliação quantitativa, não foi possível verificar qual metodologia teria solucionado o mesmo problema com menor tempo, todos eles utilizando a ferramenta.

### Estudo individual

 O estudo começou de forma individual, onde entregou-se a cada aluno o protocolo de usuário e o *framework*. Solicitou-se que os mesmos realizassem a leitura do protocolo antes de utilizar a ferramenta para que todos tivessem o discernimento para julgar e avaliar a mesma.

 O intuito de começar o estudo de forma individual, foi para que os alunos pudessem conhecer a ferramenta. Sendo assim, não foi entregue nenhum estudo de caso pronto, ou seja, o aluno deveria simular o ambiente para a sua utilização da ferramenta ou utilizá-la visando sua rotina de trabalho.

### Estudo em grupo

 No estudo em grupo, os alunos já estavam separados em 3 grupos, onde foi designado que uma equipe utilizaria a metodologia Ágil, outra equipe utilizaria metodologia Tradicional e a terceira seria livre para escolher a forma que mais lhe agradasse. A professora Adriana já havia passado um estudo de caso (Apêndice A) que se encaixaria muito bem no teste da ferramenta. Ainda, solicitou-se que cada membro do grupo tivesse um papel específico. As funções solicitadas foram:

* Analista de negócio (requisitos e modelagem);
* Gerente de projeto;
* Desenvolvedor;
* Testador;
* Manutenção/suporte.

 Observou-se que uma das maiores dificuldades é avaliar os níveis de cada um do time, onde foi tema de muita discussão nos grupos.

 O grande diferencial do estudo em grupo foi que muitas dúvidas, que surgiram na utilização do *framework* individualmente, foram esclarecidas quando a ferramenta foi utilizada em grupo, pois haviam pessoas maior conhecimento sobre o assunto. Ainda, alguns alunos leram com maior atenção o protocolo e puderam auxiliar os demais do grupo que estavam com dúvidas.

### Estudo de caso

O estudo de caso proposto por REIS (2017) é de uma empresa de agronegócio que possui um setor interno de desenvolvimento de software. Este setor tem o objetivo de entregar soluções aos demais departamentos da empresa. No estudo de caso, a empresa não possui um processo formal estabelecido e as demandas dos departamentos são tratadas como projeto ou suporte.

Ainda segundo REIS (2017),

“ No caso de projeto, um analista mapeia as necessidades do demandante, e cria uma especificação de Modelo de Casos de Uso e protótipo das interfaces gráficas. Essa tarefa não tem prazo de finalização determinado. Quando essa documentação é finalizada, então ela é passada para o Gerente de Projetos. Este atua de duas maneiras: ou ele próprio cria as atividades e determina o prazo de entrega, ou cria as atividades e solicita que os desenvolvedores estimem prazos. Algumas vezes, os desenvolvedores estimam empregando *Planning Poker*. Uma vez estimadas, o Gerente de Projetos aloca as mesmas aos desenvolvedores, e o andamento das tarefas é acompanhado em um software de controle de tarefas.

Em casos de suporte, analista de sistema ou desenvolvedor passam a demanda para o Gerente de Projetos, que determina quem dos desenvolvedores ficará 2 responsável em atendê-lo. Conforme a avaliação de urgência, o programador e/ou analista envolvido (s) para (m) suas tarefas atuais para realizar o atendimento. “

 Este estudo de caso tem o intuito de mostrar aos alunos como uma empresa trabalha e solicitar que eles corrijam o processo. Pelo estudo de caso também se observa o funcionamento de uma empresa de consultoria, que não está seguindo nenhum padrão, nem de qualidade e muito menos de Engenharia de Software.

## Questionário

 A avaliação da ferramenta foi feita através de um questionário Quanti-qualitativo, aplicado aos alunos de Engenharia de Software, com a intenção de avaliar se os objetivos do *framework* foram alcançados. Para criação do questionário, utilizou-se o Google Formulários, onde os alunos que receberam o link puderam acessar e responder as perguntas.

 As perguntas de 1 a 6 analisam a ferramenta, verificando com o usuário se a mesma está bem estruturada e indicando quais partes necessitam ajustes. Essas perguntas foram criadas com o intuito de verificar se a ferramenta poderia ser utilizada sem nenhum tipo de consultoria, ou seja, visa analisar se existe alguma dificuldade na utilização do *framework* quando utilizada por pessoas que nunca tiveram contato com a ferramenta. As perguntas de 1 a 6 são descritas da seguinte forma:

1. No protocolo do usuário, as instruções para utilização do *framework* estão claras?

1.1. Caso tenha marcado "Não" na pergunta 1, em qual parte do protocolo você ficou com dúvida?

1. No *framework*, o portfólio, na página inicial, está compreensível?
	1. Caso tenha marcado "Não" na pergunta 2, em qual parte do portfólio você ficou em dúvida?
2. Com relação ao questionário de métricas para produtos novos, as perguntas estão compreensivas?
	1. Caso tenha marcado "Não" na pergunta 3, qual pergunta não foi compreendida?
3. Com relação ao questionário de métricas para atualização, as perguntas estão compreensivas?
	1. Caso tenha marcado "Não" na pergunta 4, qual pergunta não foi compreendida?
4. Com relação às tabelas de resultados, ficou evidente o resultado e qual modelo deve ser aplicado?
	1. Caso tenha marcado "Não" na pergunta 5, em qual resultado você sentiu dificuldade de compreender?
5. Com relação a explicação de cada modelo, na tabela de resultados, estão bem explicados?
	1. Caso tenha marcado "Não" na pergunta 6, em qual explicação você sentiu dificuldade de entender?

 As perguntas 7, 12 a 14 tem o propósito de avaliar se o objetivo do *framework* foi alcançado, verificando se os resultados obtidos então de acordo com o esperado, se a ferramenta auxiliou na escolha de um modelo que melhor se encaixa na demanda do usuário e se a ferramenta trouxe algum modelo desconhecido para o avaliador. As perguntas 7, 12 a 14 estão descritas da seguinte forma:

1. Qual seu nível de satisfação após a utilização do *framework*?

12. Os resultados expressos no *framework* estão de acordo como o escopo do projeto?

 13. Implementaria a proposta de *framework* na sua empresa?

 14. Existe algum método dentro do *framework* que você desconhecia e passou a conhecer depois da utilização dessa ferramenta?

 14.1 Caso tenha marcado "Sim" na pergunta 14, qual você desconhecia?

 As perguntas de 8 a 11 avaliam o usuário, questionando se o mesmo compreende os métodos e metodologias abordados no *framework* a fim de verificar se o mesmo possui algum conhecimento prévio em relação as metodologias abordadas. As perguntas de 8 a 11 estão descritas da seguinte forma:

1. Você trabalha com desenvolvimento de software?
2. Você trabalha em uma empresa que utiliza alguma metodologia? Se sim, qual?
3. Qual seu conhecimento com relação as metodologias Tradicionais?
4. Qual seu conhecimento com relação as metodologias Ágeis?

### Avaliação das perguntas

 Na avaliação dos alunos, obtivemos os mais variados resultados, devido as diversidades de empresas em que os usuários atuam, seus conhecimentos quanto as metodologias e conhecimento em Engenharia de Software.

 O principal objetivo da ferramenta é designar o método mais apropriado a ser aplicado. Segundo a maioria dos alunos, este objetivo foi alcançado, porém, constatou-se através da avaliação dos alunos que a ferramenta precisaria de alguns ajustes para ser utilizado de forma efetiva.

 Analisando as respostas da pergunta 1, 81,3% dos alunos não sentiram dificuldade em interpretar o texto descrito no protocolo. Foi sugerido que solicitasse a leitura do protocolo dentro do *framework*, podendo, em uma próxima versão, introduzir a solicitação da leitura do protocolo no portfólio.

 Na pergunta 2, 93,8% dos alunos afirmaram que o portfólio na página principal do *framework* está compreensível. Sendo assim, verificamos que o portfólio não precisa de grandes ajustes, apenas acrescentando a solicitação da primeira pergunta.

 62,5 % dos alunos compreenderam as questões abordadas em métricas para produtos novos, segundo a pergunta 3. O restante dos alunos sentiu dificuldade em classificar as perguntas em níveis de 1 a 5, sugerindo que algumas dessas perguntas fossem respondidas com “sim” ou “não”. Não comentaram quais perguntas deveriam ser respondidas sem serem classificadas por níveis.

 Durante a avaliação individual, os alunos puderam analisar as perguntas referentes a métricas para atualização, porém, quando o *framework* foi utilizado em grupo, nenhum deles optou pelas métricas para atualização. Desta forma, na questão 4, 81,3% dos alunos afirmaram que as perguntas em métricas para atualização estão compreensíveis, porém comentaram que não utilizaram esta métrica.

 A questão 5 está questionando sobre o resultado que a ferramenta expôs, se está claro qual modelo deve ser aplicado. 75% dos alunos afirmaram que os resultados da ferramenta estão sendo executados corretamente, porém, necessita-se alguns ajustes para facilitar a visualização do usuário, deixando em maior evidência o método a ser aplicado. Ainda referente aos resultados, temos a questão 6, onde 100% dos alunos responderam que a descrição de cada modelo está bem explicado.

 Na pergunta 7 verificou-se que 12,5% dos usuários sentiram-se muito satisfeitos após a utilização da ferramenta e 81,3% dos usuários estão satisfeitos, ou seja, a ferramenta agradou 93,8% dos usuários. Apenas um usuário da ferramenta sentiu-se insatisfeito após a utilização da mesma.

 As perguntas 8, 9,10 e 11 buscam conhecer o usuário. O que é fundamental para traçar um perfil das pessoas que responderam o questionário, analisando o seu conhecimento na área de Engenharia de Software e verificando se pessoas que não participam diariamente da construção de software seriam capazes de utilizar a ferramenta com a mesma eficácia de quem conhece as técnicas. Verificou-se através da questão 8 que 56,2% dos usuários já trabalharam com desenvolvimento de software, sendo relevante para um melhor compreendimento da ferramenta. A questão 9 revelou que 50% das empresas dos usuários não utilizam nenhuma metodologia em seus projetos, desta forma, a ferramenta poderia trazer algum benefício a essas empresas, indicando um método adequado a suas demandas. A questão 10 mostra que 56,3% dos alunos não conhece quase nada referente as metodologias tradicionais e que o restante dos alunos conhece algumas dessas metodologias tradicionais. Já na questão 11, 75% dos alunos conhece algumas das metodologias ágeis, apenas um indivíduo conhece todas as metodologias e o restante dos alunos não conhece quase nada das metodologias ágeis. Desta forma, através das respostas das questões 10 e 11 verifica-se que as metodologias ágeis são mais conhecidas que as tradicionais, devido a estarem em maior evidência atualmente.

 Na pergunta 12 analisamos se os resultados do *framework* estão de acordo com o escopo do projeto e segundo as respostas dos usuários, 81,3% concorda com os resultados.

 A pergunta 13 questiona os usuários se eles implementariam em suas empresas o *framework*. Apenas um dos usuários afirma que com certeza implementaria a ferramenta em sua empresa, porém, 68,8% respondeu que implementaria a ferramenta dentro da empresa, porém, a empresa poderia não aceitar. Desta forma, observa-se que muitas empresas podem ainda ter uma visão tradicional e podem não estar abertas a mudanças.

 A pergunta 14 e 14.1 questionam se há na ferramenta algum método desconhecido pelos usuários e qual ele passou a conhecer utilizando a ferramenta. 37,5% dos usuários afirmam que desconheciam alguns dos métodos apresentados no *framework*, sendo o JAD o mais desconhecido pelos usuários. 62,5% conheciam todos os métodos abordados pela ferramenta.

## Considerações sobre a avaliação do *framework*

 O estudo piloto foi extremamente importante para julgar a ferramenta, onde os alunos puderam fazer críticas, elogios e sugestões. Esse estudo tem o intuito de verificar se o objetivo da ferramenta foi alcançado, que era auxiliar os usuários na escolha de quais as técnicas mais apropriadas para aplicar dentro de um projeto de desenvolvimento de software.

 As avaliações do *framework* de forma geral foram positivas e manteve os usuários satisfeitos com os resultados e com a experiência de utilizar o *framework*. Porém, ainda é necessário deixar a ferramenta mais intuitiva, de forma a ser utilizada por qualquer pessoa, sem maiores dificuldades.

 Analisando as respostas dos alunos, podemos concluir que muitas empresas não possuem um modelo específico de construção de software, mesmo tendo funcionários com conhecimento de tais práticas. As companhias mantêm-se com a mente fechada para novas técnicas, dificultando a aceitação para mudanças, o que se torna um empecilho para aplicação da ferramenta no mundo empresarial.

CONCLUSÃO

 Com todo o avanço da tecnologia, a Engenharia de Software não pode ficar parada no tempo. Desta forma, está em constante evolução, buscando novas maneiras de aumentar o rendimento e a qualidade na produção de software*.* Os métodos Tradicionais foram os pioneiros no desenvolvimento do processo de software, possuindo pontos positivos como serem de fácil compreensão e implementação, reforçam bons hábitos e funcionam bem com times frágeis. Porém, devido ao grande volume de documentação para sua implementação, se tornaram obsoletas. Atualmente, podemos dizer que as metodologias ágeis estão em evidência, pois produzem apenas o necessário de documentos, buscando evitar desperdício de tempo, priorizando a comunicação pessoal, onde o diálogo vem sempre em primeiro lugar e os processos são bem definidos.

 Ao longo da busca de referências bibliográficas ficou evidente a carência existente de uma metodologia que otimizasse a equipe, que analisasse a maturidade das empresas, o escopo dos projetos e as equipes de desenvolvimento. Através dessa necessidade, este estudo buscou revisar os métodos existes na literatura de requisitos, modelagem de software, construção, testes e manutenção em busca de estruturar os métodos e, após, foi construído um *framework* que seja possível utilizar o método mais adequado para cada projeto, equipe e empresa.

 A construção de um *framework* que seja adaptável conforme a necessidade do projeto é extremamente relevante para o mundo acadêmico e empresarial, pois encontramos diversos modos de construir um software. Porém, geralmente as empresas aderem a um modelo específico, ou ainda pior, não aderem a modelo de processo algum. Não avaliam se sua equipe irá aderir ao método ou se o modelo para o projeto é adequado e se trará resultados positivos a empresa.

 O *framework* foi elaborado, a partir de muito estudo, pois não foi encontrado na literatura algum autor que conceituasse a dificuldade de implementação de cada método, sendo necessário a revisão e leitura de todos os métodos estudados para construir a ferramenta. Também, a partir desse estudo, foi possível elaborar o cálculo que traz o resultado de cada item do ciclo de vida na construção de um processo de software*.*

 Analisando os resultados expressos pelos alunos, é possível identificar alguns pontos que são passíveis de ajustes, para facilitar a utilização da ferramenta. De modo geral, a utilização do *framework* foi bem vista pela avaliação dos alunos, onde muitos demonstraram interesse em utilizar a ferramenta em suas empresas, porém, ainda teriam grande resistência na implementação nas empresas. Ainda sobre a avaliação, é possível afirmar que o principal objetivo, que é auxiliar na tomada de decisão de quais as técnicas mais apropriadas para aplicar dentro de um projeto de desenvolvimento de software, foi alcançado, onde a maioria dos usuários concorda com os resultados expressos pela ferramenta.

 Os objetivos do trabalho foram alcançados, onde dentro deste documento foram mapeados e analisados todos os métodos existentes na literatura, separando-os pelo ciclo de vida na construção de um software. Identificou-se os pontos positivos e negativos das metodologias, os quais estão descritos ao longo deste trabalho. Organizou-se um *framework* com a composição de processos híbridos, avaliou-se dentro do *framework* a avaliação de contextos reais para sua adoção e a ferramenta propõem recomendações a respeito de quais práticas são as mais adequadas, analisando o contexto expresso pelo usuário da mesma.

 Mesmo com os objetivos deste trabalho sendo alcançados, é necessário um estudo mais aprofundado para montar uma análise quantitativa, ou seja, se a ferramenta trará um benefício ou não a Engenharia de Software.

 Verificou-se que alguns ajustes são necessários para a melhoria da ferramenta. É interessante colocar a solicitação de leitura do protocolo de usuário no portfólio da ferramenta, para que todos os usuários leiam com atenção o protocolo a fim de utilizarem corretamente a mesma. Através das respostas dos alunos, observou-se que seria interessante demostrar os resultados de forma mais clara, como por exemplo, os resultados de teste deveriam ser utilizados as cores da sinaleira, indicando até qual ponto o usuário deve parar de realizar os testes. Dentro do questionário a pergunta 7 onde questionamos se o usuário está satisfeito após a utilização do *framework*, deveria questionar o usuário quais pontos sentiu-se insatisfeito e o motivo de sua insatisfação.

 Um estudo mais aprofundado seria ideal para montar uma análise quantitativa dos resultados, analisando se a ferramenta traria benefícios ou não a Engenharia de Software. Porém é possível concluir, através das respostas da pergunta 13, que encontraríamos dificuldades para utilizar o *framework* no mundo empresarial.

 Concluiu-se com este estudo que a ferramenta tem potencial para auxiliar as equipes na escolha de um método adequado. Para podermos avaliar a eficiência efetiva do *framework*, seria ideal empregar a ferramenta dentro de uma empresa e realizar uma análise mais precisa da mesma.

Referências Bibliográficas

AGILE MANIFESTO. **Manifesto for Agile Software Development.** Disponível em: < http://agilemanifesto.org/> Acesso em: 03 set. 2016

AURUM, A.; WOHLIN, C. **Engineering and Managing Software Requirements.** 1ª Ed. Sydney, Australia. Editora Springer, 2005.

BATISTA, E. A.; CARVALHO, A. M. B. R. Uma Taxonomia Facetada para Técnicas de Elicitação de Requisitos. 2003.

BECK, P.; JIANG, J. J.; KLEIN, G. **Prototyping mediators to project: Learning and interaction.** The Journal of Systems and Software. p. 1025 – 1035. 2006.

BEZERRA, E. **Princípios da análise e projetos de sistema com UML**. 3ªEd. Rio de Janeiro. Editora Campus, 2006.

BLANKENSHIP, J.; BUSSA, M.; MILLETT, S. **Pro Agile .NET development with scrum.** New York, 2011.

BOEHM, B. The Spiral Model as a Tool for Evolutionary Software Acquisition. Best Practice. 2001.

CHEN, P. The Entity-Relationship Model – Roward a Unified View of Data. Massachusetts Institute of Technology, 1976.

DEVMEDIA. **Engenharia de software 2 – Técnicas para levantamento de requisitos**. Disponível em: <<http://www.devmedia.com.br/engenharia-de-software-2-tecnicas-para-levantamento-de-requisitos/9151>>. Acesso em: 29 out. 2016.

FILHO, W. de P. P. Engenharia de Software: fundamentos, métodos e padrões. Editora LTC. p. 260. 2000.

FOGGETTI, C. **Gestão ágil de projetos.** São Paulo, SP. Editora Pearson, 2015.

GAMMA, E. et al. Design Patterns Elements of Reusable Object-Oriented Software. 1997.

GANE, C.; SARSON, T. **Analisis Estructurado de Sistemas.** Buenos Aires, Editora El Ateneo, 1987.

KENDALL, K.E.; KENDALL, J.E.. **Sistems Analysis and Design**. 8ª Ed. New Jersey. Editora Pearson, 2011.

LINHA DE CÓDIGO. **A importância do levantamento de requisitos no sucesso dos processos de software.** Disponível em: <<http://www.linhadecodigo.com.br/artigo/1685/a-importancia-do-levantamento-de-requisitos-no-sucesso-dos-projetos-de-software.aspx>>. Acesso em: 29 out. 2016.

MUNASSAR, N. M. A.; GOVARDHAN, A. **A comparison between five models of software engineering.** IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September 2010.

NATARAJAN, H., SOMASUNDARAM, R. K., LAKSHMI, K. **A comparison between presente and future models of software engineering.** IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 2. 2013.

NATIONAL INSTRUMENTS. **Development life cycle models.** <<http://zone.ni.com/reference/en-XX/help/371361N-/lvdevconcepts/lifecycle_models/>> Acesso em: 25 out. 2016.

OLIVEIRA, L. R. de. **Implementação de processos: uso de técnicas de estimativas de projetos de software para estimar processos de negócios.** Projeto de dissertação submetido para qualificação no curso de mestrado profissional em sistemas de informação e gestão do conhecimento da Faculdade de Ciências Empresariais da Universidade FUMEC. Belo Horizonte, 2012.

PRESSMAN, R.S. **Engenharia de Software: Uma Abordagem Profissional**. 8. ed. São Paulo, SP: McGraw-Hill, 2016.

PRODANOV, C. C.; FREITAS, E.C.. Metodologia do trabalho científico: Métodos e técnicas da pesquisa e do trabalho acadêmico. 2ª Ed. Novo Hamburgo, RS: Feevale, 2013.

RUMBAUGH, J. JACOBSON, I.; BOOCH, G. **The Unified Modeling Language Reference Manual.** 2ª Ed. Editora Pearson, 2005.

REIS, A. N. **Estudo de caso: Processo de Desenvolvimento**. Novo Hamburgo, RS: Feevale, 2017.

REZENDE, D. A. **Engenharia de Software e** **Sistemas de Informação.** 3ª Ed. Editora Brasport, 2005.

SCHWABER, K., SUTHERLAND, J. **The Scrum GuideTM. The definitive guide to scrum: the rules of the game.** Disponível em: < <http://www.scrumguides.org/>>. Acesso em: nov. 2016.

SILVA, J. B.; SAMPAIO, M.; PEZZIN, J. Usando Ontologias na Construção de Modelos MDA (MODEL-DRIVEN ARCHITECTURE). Salvado, BA. 2006.

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

SVERRISDOTTIRA, H.S., INGASONB, H. T., JONASSONC, H. I. **The role of the product owner in scrum - comparison between theory and practices.** Procedia - Social and Behavioral Sciences 119 p. 257 – 267. 2014.

WAZLAWICK, R. S. **Engenharia de Software Conceitos e práticas.** 1ª Ed. Rio de Janeiro, RJ. Editora Elsevier, 2013.

**ANEXO A – Protocolo do usuário do *framework***

 Esta ferramenta foi desenvolvida em caráter de avaliação do trabalho de conclusão de Lucas Eduardo Hartz, com objetivo de auxiliar o usuário na escolha de um método para desenvolver um software no menor tempo possível, visando: a equipe disponível, tempo disponível para conclusão do projeto, a maturidade da empresa desenvolvedora e da empresa contratante.

 Grande parte das empresas se preocupa apenas com a metodologia empregada, geralmente a tradicional ou ágil. Visando quebrar esse paradigma, estudou-se os métodos existentes na literatura, unindo-os em uma única ferramenta, a fim de auxiliar a equipe na escolha do melhor modelo, independente da metodologia escolhida.

 O *framework* foi construído a partir da literatura, podendo haver métodos não documentados que não estão listados dentro da ferramenta.

 O f*ramework* foi dividido em dois situações, um para desenvolvimento de software novos e outro para atualizações. A utilização deve ser feita antes do início do projeto, onde existe apenas a necessidade do software e a equipe que irá participar do projeto. Após, não seria viável a utilização da mesma.

 Deve ser preenchido um dos questionários com o máximo de precisão possível, ou de produtos novos ou de atualizações, conforme a necessidade. Após avalia-se os resultados.

 A tabela avalia a complexidade e grau de dificuldade de aplicar os métodos, portanto não implica que o método sugerido seja o melhor para sua demanda.

 A leitura dos resultados de requisitos, modelagem e manutenção é bastante simples. No console de resultados, onde está apontando uma média ponderada, apresentará um valor no intervalo de 1 a 5 e as cores da sinaleira (verde, amarelo e vermelho), sendo os valores de 1 para a cor verde, 2,5 para amarelo e 5 para o vermelho. Ao lado de cada modelo há um número entre parênteses, indicando que no resultado da média ponderada irá mostrar qual o modelo indicado para utilização. O mesmo se aplica as cores, que servem para facilitar a visualização dos resultados. Nos resultados de teste é diferente, pois aconselha-se que os testes sejam feitos até o número indicado, ou seja, deve-se fazer todos os modelos até o valor que está na média.

 Existe uma breve explicação do funcionamento de cada método, basta parar o mouse em cima do modelo e ele lhe mostrará em forma de lembrete.

 Após a utilização da tabela, será necessário o preenchimento de um questionário para avaliação do *framework*, mostrando se foi possível ganhar tempo na utilização da ferramenta.

**APÊNDICE A – Estudo de caso**

**Engenharia de Software II**

*Prof.ª Adriana Neves dos Reis*

**Estudo de Caso:** Processo de Desenvolvimento

Uma equipe de desenvolvimento de software faz parte de uma organização como um setor de entrega de soluções para os demais departamentos da empresa, os quais estão diretamente ligados aos seus serviços de projetos e análises químicas para o agronegócio.

Para isso, ela conta com:

* 2 analistas de negócio;
* 1 gerente de projeto;
* 2 analistas de sistemas;
* 3 desenvolvedores C#;

Para o fluxo de trabalho, não há um processo formal estabelecido. Normalmente, os diferentes departamentos da empresa geram demandas que são tratadas como **projeto** ou **suporte**.

No caso de projeto, um analista mapeia as necessidades do demandante, e cria uma especificação de Modelo de Casos de Uso e protótipo das interfaces gráficas. Essa tarefa não tem prazo de finalização determinado. Quando essa documentação é finalizada, então ela é passada para o Gerente de Projetos. Este atua de duas maneiras: ou ele próprio cria as atividades e determina o prazo de entrega, ou cria as atividades e solicita que os desenvolvedores estimem prazos. Algumas vezes, os desenvolvedores estimam empregando *Planning Poker*. Uma vez estimadas, o Gerente de Projetos aloca as mesmas aos desenvolvedores, e o andamento das tarefas é acompanhado em um software de controle de tarefas.

Em casos de suporte, analista de sistema ou desenvolvedor passam a demanda para o Gerente de Projetos, que determina quem dos desenvolvedores ficará 2responsável em atendê-lo. Conforme a avaliação de urgência, o programador e/ou analista envolvido(s) para(m) suas tarefas atuais para realizar o atendimento.

Ao finalizar a tarefa, o desenvolvedor a sinaliza na ferramenta com o estado “em teste”. O analista então realiza os testes, sem nenhum tipo de especificação. Caso não encontre problemas, o analista muda o estado da tarefa para “concluída”. Caso encontre erros, o analista muda o estado da tarefa para “em desenvolvimento”. Esses estados são acompanhados pela opção de histórico da ferramenta de controle de tarefas.

Independente de projeto ou suporte, uma vez finalizado, é solicitado que o demandante realize o teste final. Quando esse for finalizado, então a solução entregue passa a ser usada (pela própria empresa), implantada (para um cliente específico), ou comercializada (um produto).

Os analistas de negócio atuam como consultores da equipe em qualquer momento para esclarecer dúvidas relacionadas ao ramo de atuação da empresa.

Apesar de ser uma equipe experiente (todos atuam há mais de 2 anos na empresa), os desenvolvedores reclamam da inexistência de um padrão de fluxo de trabalho. Além disso, em algumas situações de acúmulo de tarefas, o ambiente de trabalho se torna pouco saudável, e há muitos desentendimentos no que diz respeito à responsabilidade e comprometimento de todos da equipe.