ENGENHARIA DE SOFTWARE

NOSSA HISTÓRIA

A nossa história inicia-se com a ideia visionária e da realização do sonho de um grupo de empresários na busca de atender à crescente demanda de cursos de Graduação e Pós-Graduação. E assim foi criado o Instituto, como uma entidade capaz de oferecer serviços educacionais em nível superior.

O Instituto tem como objetivo formar cidadão nas diferentes áreas de conhecimento, aptos para a inserção em diversos setores profissionais e para a participação no desenvolvimento da sociedade brasileira, e assim, colaborar na sua formação continuada. Também promover a divulgação de conhecimentos científicos, técnicos e culturais, que constituem patrimônio da humanidade, transmitindo e propagando os saberes através do ensino, utilizando-se de publicações e/ou outras normas de comunicação.

Tem como missão oferecer qualidade de ensino, conhecimento e cultura, de forma confiável e eficiente, para que o aluno tenha oportunidade de construir uma base profissional e ética, primando sempre pela inovação tecnológica, excelência no atendimento e valor do serviço oferecido. E dessa forma, conquistar o espaço de uma das instituições modelo no país na oferta de cursos de qualidade.

Sumário

1. PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

1.1 Definição e importância processo de desenvolvimento de software

Um software não se constitui unicamente pelas linhas de código executáveis. Incorpora toda a documentação necessária para a sua instalação, uso e manutenção, incluindo a configuração imprescindível ao seu correto funcionamento.

Diante deste entendimento, o Processo de Desenvolvimento de Software ou simplesmente Processo de Software, se refere a toda uma gama de ações executadas no âmbito do desenvolvimento de um produto de software, incorporando desde o levantamento de requisitos, escolha da linguagem, e alcançando inclusive, a análise de performance (SOMMERVILLE, 2011, HIRAMA, 2012; PETERS, 2001).

Assim, no Processo de Software são estabelecidos os métodos (norteia o que fazer), as ferramentas (define o material a ser utilizado), e os procedimentos (indicando o como fazer) de modo a garantir que o produto alcance o resultado planejado (PRESSMAN, 2011).

É justamente pelo desenvolvimento de um bom Processo de Software que se pretende alcançar a qualidade do produto desenvolvido e/ou evoluído, o que por sua vez requer organização e definição de ações atuantes para o alcance de agilidade e transformação tecnológica, considerando o contexto de prazo e assertividade em produto (WAZLAWICK, 2013).

Tratando destes aspectos, Wazlawick (2013) segue destacando que o prazo incorpora o tempo de execução do projeto, incluindo os limites e datas estabelecidos previamente, e a assertividade em produto se faz relacionada ao fato de o software, uma vez entregue ao cliente, se fazer preparado para atender as demandas que lhe foram solicitadas sem a incorrência de falhas ou a necessidade frequente de paralisação para manutenções e correções que não foram previamente definidas.

Tal preocupação, deu-se a partir anos de 1960, quando, por falta de padronização no desenvolvimento, a demanda crescente de desenvolvimento, os custos e prazos mal definidos, o serviço descompromissado das equipes de tecnologia e a ausência de confiança dos clientes, fez surgir uma fase denominada pela expressão Crise do Software, evidenciando a necessidade de que adequações fossem definidas para que o processo se tornasse mais profissional (SOMMERVILLE, 2011).

Era destaque à época a questão referente aos defeitos de software em face de seu mal dimensionamento, o custo elevado e a estimativa de prazos que não conseguia se fazer cumprida. A isto, somava-se a complexidade do código e a total ausência de documentação, o que dificultava a manutenção, atrapalhando o processo de evolução do software e o consequente acompanhamento ao progresso dos equipamentos existentes (PRESSMAN, 2011).

Além deste fato, a comunicação entre o cliente e a TI era precária ou inexistente, acarretando uma série de falhas no desenvolvimento pelo fato de a equipe de tecnologia ter dificuldade em compreender as reais necessidades a serem atendidas pelos softwares, levando a correções praticamente infinitas (WAZLAWICK, 2013).

Outro aspecto impactante era a ausência de meios avaliativos de eficácia que permitissem proceder com a definição de estimativas de maior precisão em relação ao software desenvolvido e entregue, o que efetivamente impactava na possibilidade de se prospectar a necessidade de treinamentos e de manutenções preventivas aos sistemas (SOMMERVILLE, 2011).

Assim, a Crise do Software impulsionou o Processo de Software, e este, por sua vez, passou a englobar atividades fundamentais voltadas ao desenvolvimento e/ou evolução em um processo de software, as quais seguem apresentadas no Quadro 1.

Atividades fundamentais para o desenvolvimento e/ou evolução do processo de software

Especificação de software	Atividade em que se define as funcionalidades que devem ser atendidas pelo software e possíveis restrições que podem ser interpostas ao projeto.
Projeto e Implementação de software	Atividade longa que parte das especificações definidas anteriormente e se volta ao desenvolvimento do software.
Validação de Software	Atividade de validação da ferramenta desenvolvida, no intuito de se atestar se a solução realiza o que fora solicitado pelo cliente.
Evolução de software	Atividade de ajuste com mudanças voltadas a garantir total atendimento às necessidades do cliente.

Quadro: 1

Tais ações demosntram o nível de profissionalismo alcançado a partir do Processo de Software em oposição à ausência de definição de processos como acontecia anteriormente à Crise do Software.

O Processo de Software passou a permitir efetivas melhorias em relação ao treinamento de usuários, padronização de desenvolvimento e possibilidade da experiência do usuário ser mais bem compreendida e aperfeiçoada por meio do sistema (WASTLAWICK, 2013).

Neste âmbito, cabe reconhecer que quando um Processo de Software é especificado também são definidas descrições como as referenciadas no Quadro 2.

Outras especificações importantes no Processo de software

Produtos	Resultado que se deseja alcançar com o processo que se encontra em execução.
Papéis	Pessoas responsáveis e envolvidas no processo.
Condições	Declarações prévias ou posteriores ao processo e que merecem atenção

Quadro: 2

Cabe destacar em relação ao Quadro 2, que quando definidos os papeis referentes às pessoas responsáveis e envolvidas no processo tem-se um termo técnico comumente utilizado que é o de Stakeholders, ou seja, todos aqueles que são responsáveis, possuem interesses e/ou ainda são afetados pela criação

de softwares e pela incorporação deles nas atividades de negócio (OLIVEIRA, 2018).

No intuito de que o Processo de Software, embasado nas ações e especificações já apontadas, alcançasse a possibilidade de acompanhar o desenvolvimento e a evolução de um software, concebeu- se o entendimento acerca do Ciclo de Vida de Software (WAZLAWICK, 2013).

O Ciclo de Vida de Software, de acordo com Wazlawick (2013) incorpora em si as ações pertinentes a todo o período de existência de um software, incluindo a atualização do sistema em face das atividades do usuários sofrerem ajustes, o que, por conseguinte, leva a alterações evolutivas na ferramenta.

1.2 Modelos de processos de software

Um Processo de Software configura-se em um conjunto de atividades complexas e especializadas, de modo que o Modelo de um Processo de Software representa e fornece, de modo simples, informações e visibilidade acerca do processo em si traduzidas na forma de estruturas/esquematizações fornecedores de uma visão didática de como proceder com as etapas que devem se fazer efetivadas (SOMMERVILLE, 2011; PRESSMAN, 2011).

Um Modelo de Processo de Software possui em seu bojo estruturas que demonstram as etapas atinentes ao desenvolvimento do software em si, por meio da organização do Ciclo de Vida de Software, no qual tem-se estruturadas as atividades a serem executadas, incorporando em si os conceitos listados no Quadro 3.

Ciclo de Vida de Software

Requisitos	Etapa voltada ao levantamento de informações atinentes ao problema que a ferramenta de software se destinará a resolver. Tais informações, ou seja, os requisitos configuram-se como condições, necessidades, características, funcionalidades, restrições e demais aspectos que carecem ser atendidos, necessitando ser devidamente documentados.
Análise	Etapa que parte do levantamento de requisitos e permite que a equipe de processo conheça a atividade do

	usuário, discuta os requisitos levantados e identifique melhorias que carecem de implementação.
Abstração e representação	Etapa de elaboração de modelos mentais e conceituais voltados a solucionar as necessidades do cliente por meio do software.
Projeto	Etapa de desenho do produto de software, partindo dos requisitos, ou seja, do que o sistema deverá executar.
Implementação	Etapa de transformação do projeto através da linguagem de programação escolhida, não devendo se fazer completamente dissociada da interação com o cliente.
Integração	Etapa em que os módulos desenvolvidos durante a implementação se fazem integradas para que possam trocar dados entre si.
Testes	Etapa de verificação quanto ao alcance de objetivos por parte do software e da possibilidade de falhas de execução ou funcionalidades mal dimensionadas.
Implantação	Fase em que estando o sistema finalizado, ele é disseminado pela organização e o desenvolvedor se mantém em atenção para a necessidade de ajustes em face de questões de usuário ou modelo de negócios.
Manutenção	Fase em que após a entrega, faz-se importante a correção de falhas, a melhoria de desempenho e adaptações de acordo com a necessidade do cliente.

Quadro: 3

Cabe, por último considerar a importância da documentação completa do Processo de Software, no qual efetivamente, através de diagramas de casos de uso especifica-se todo o ciclo, incluindo desde os requisitos até o planejamento de manutenção, garantindo que outras equipes de tecnologia possam tranquilamente se posicionar em prol da evolução do sistema.

Em relação à fase de Implantação apresentada a partir do Quadro 3, deve-se considerar os seguintes possibilidades descritas no Quadro 4 quando um software se fizer aplicado em substituição a outro que esteja em funcionamento.

Direita	Assim que o sistema novo entrar em pleno funcionamento o antigo é retirado de operação completamente.
Paralela	Os sistemas, novo e antigo, permanecem funcionando paralelamente com acesso e atualização à mesma base durante certo espaço de tempo.
Piloto	O sistema novo refaz os processos efetivados pelo antigo com vistas a identificar os parâmetros de seu funcionamento e efetividade.
Parcial	Parte dos sistemas se complementam e aos poucos o novo sistema vai tomando todo o espaço do antigo.

Quadro: 4

Os conceitos apresentados apenas clarificam em relação às suas funções, e que não existe um único modelo para o desenvolvimento de um projeto, cabendo a escolha por parte da equipe de desenvolvimento embasada na filosofia de trabalho que emprega (PRESSMAN, 2011).

No entanto, a possibilidade de associação entre modelos é uma realidade, de modo a garantir que se combine os melhores aspectos entre eles (SOMMERVILLE, 2011).

1.2.1 Modelo em Cascata

O Modelo em Cascata é clássico, desenvolvendo-se de forma sistemática, encadeada e previamente planejada em sua totalidade, de modo que uma etapa depende diretamente da outra e a próxima somente se inicia a partir do momento que a anterior se encontra finalizada (SOMMERVILLE, 2011; PRESSMAN, 2011).

A Figura 1 dispõe representação clássica do Modelo em Cascata permitindo melhor entendimento de suas fases.

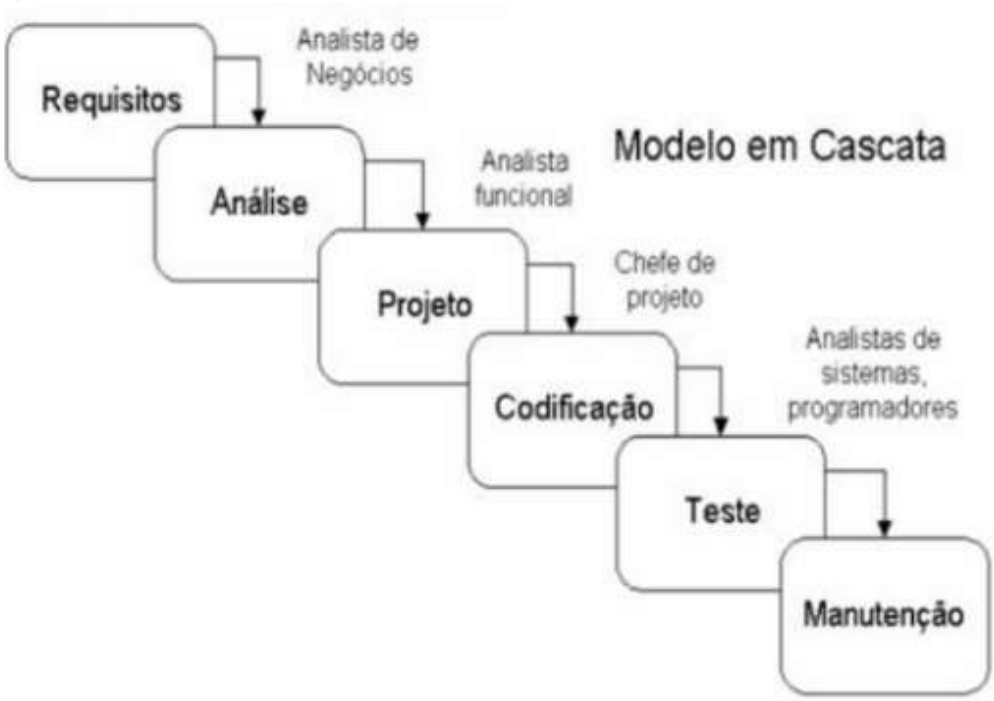

Figura: 1
Representação dos estágios do Modelo em Cascata

Justamente pelo fato de este modelo seguir um planejamento prévio, também chegou a receber a denominação de Processo dirigido a Planos, no entanto, vale a crítica de que, mesmo em sua orientação a planos, por vezes o modelo efetiva levantamento de requisitos de modo superficial, não define processos padronizados a serem seguidos pela equipe como um todo e conta com certa dificuldade em rever as falhas antes da finalização de uma etapa, e até do projeto como um todo (SOMMERVILLE, 2011; PRESSMAN; MAXIM, 2016).

Assim, pela não ocorrência de critérios de avaliação e controle que favoreçam o feedback das ações efetivadas durante a execução do projeto, tem-se que, por vezes, o projeto consegue ser paralisado em uma dada etapa em vista de conseguir cumprir os critérios estabelecidos, levando à não obediências de prazos ajustados com o cliente e, por conseguinte, o aumento do custo final do produto (PRESSMAN, 2011).

Para este modelo a documentação se faz desenvolvida a cada fase, permitindo que as ações sejam visíveis para a gestão, no entanto a possibilidade de replanejamento e mudanças durante a execução não é parte do seu escopo,

demonstrando riscos e incapacidades de gestão latentes (PRESSMAN, 2011; PRESSMAN; MAXIM, 2016).

No entanto, cabe ressaltar que, em caso de definições bem efetivadas de requisitos e ausência de necessidade de mudanças/ajustes ao longo do desenvolvimento do sistema, o modelo se aplica perfeitamente, principalmente em face de sua simplicidade de compreensão e uso (WAZLAWICK, 2013).

1.2.2 Modelo em Espiral

Neste modelo, o processo se faz representado por uma espiral, devendo se fazer percorrido no sentido horário, de dentro para fora, no intuito de demonstrar que a evolução é a tônica do conjunto de ações efetivadas (SOMMERVILLE, 2011; PRESSMAN, 2011).

Os riscos referentes ao não cumprimento de prazos, à mudança de tecnologia e ao custo com dimensionamento mal efetivado tornam-se gerenciáveis, permitindo que ajustes sejam passíveis de efetivação antes da finalização do produto (PRESSMAN, 2011; PRESSMAN; MAXIM, 2016). A Figura 2 dispõe representação do Modelo em Espiral permitindo melhor entendimento de suas fases.

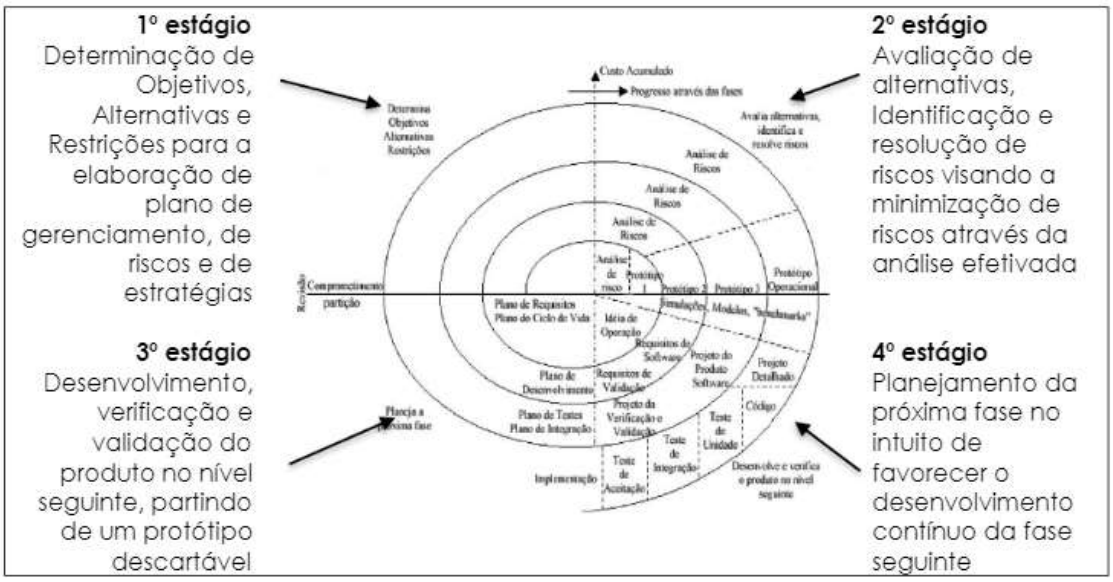

Figura: 2
Representação dos estágios do Modelo em Espiral

Conforme se observa na figura, à medida que se efetiva o desenvolvimento do projeto, cada fase adiciona novos requisitos, passando pelos estágios mais de uma vez até que o produto esteja finalizado, garantindo que os riscos sejam o fator de maior atenção no modelo, assim como a qualidade.

1.2.3 Modelo em Prototipação

O protótipo, sendo uma representação visual em baixa fidelidade do produto, permite ao usuário/cliente conhecer a proposição das funcionalidades efetivadas pela equipe de desenvolvimento através da experimentação, respaldando inclusive a troca de informações com vistas à efetivação de melhorias para a versão final (SOMMERVILLE, 2011; PRESSMAN, 2011; PRESSMAN; MAXIM, 2016).

Contudo, o Modelo em Prototipação é indicado nos casos em que o cliente não consegue identificar claramente suas necessidades em matéria de software, mas participa ativamente das etapas do projeto, e assim, por meio da interação com o produto proposto consegue estabelecer as funcionalidades de que necessita com maior precisão (SOMMERVILLE, 2011).

Seu custo que tende a ser relativamente baixo, demonstra atratividade pela incorporação de protótipos aos processos de desenvolvimento, uma vez que o levantamento de requisitos se torna mais eficiente, o feedback do cliente/usuário permite ajustes antes da finalização do produto e a usabilidade pode ser facilmente testada e ajustada (PRESSMAN, 2011).

Assim, o modelo se aplica plenamente para o levantamento e validação de requisitos, como também para a avaliação de projetos e erros de interface, permitindo que a experiência do usuário seja o diferencial desse processo (WAZLAWICK, 2013).

A Figura 3 dispõe representação de Modelo em Prototipação permitindo melhor entendimento de suas fases.

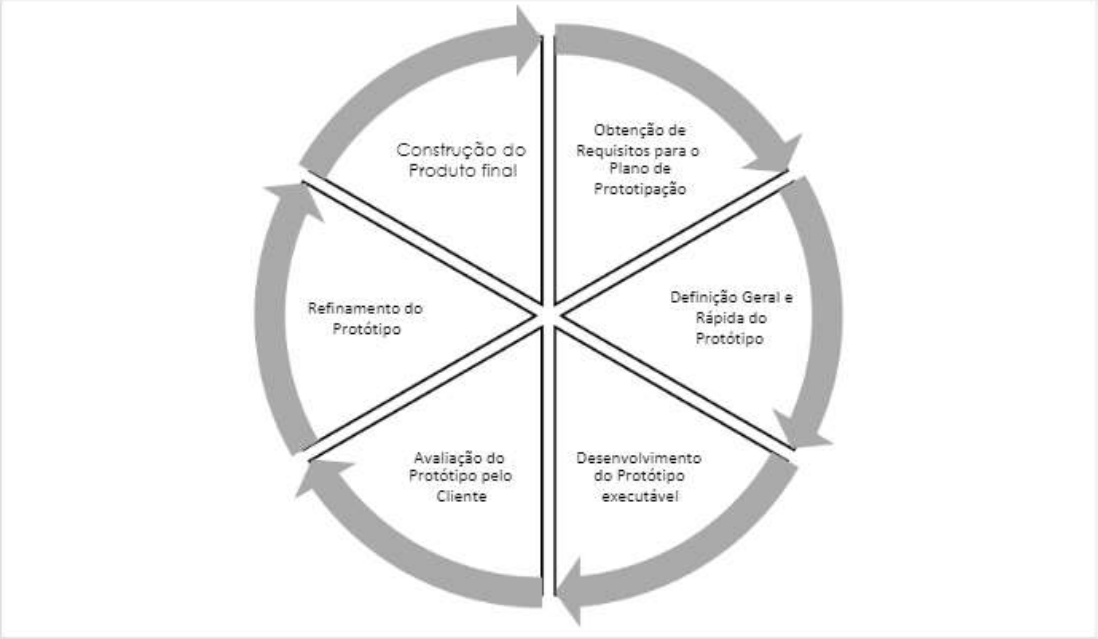

Figura: 3
Representação dos estágios do Modelo em Prototipação

Com sua etapa inicial na etapa de Obtenção de Requisitos para o Plano de Prototipação, observa-se que em seu processo cíclico melhorias podem ser incorporadas até que o protótipo alcance a possibilidade de atender às necessidades do cliente, permitindo que o produto alcance condições de efetivamente implementado em código.

Assim, compreende-se a partir de sua atuação como um objeto de experimentação, que o protótipo permite a adição e teste de funcionalidades, para que, diante de sua viabilidade, possam efetivamente fazer parte do produto em si (SOMMERVILLE, 2011).

1.2.4 Modelo em Desenvolvimento Incremental

No Modelo em Desenvolvimento incremental a criação do sistema ocorre em versões. A cada versão ajustes são adicionadas após terem passado por processo de avaliação efetivado por parte dos clientes/usuários (SOMMERVILLE, 2011; PRESSMAN; MAXIM, 2016).

Assim, o desenvolvimento ocorre em ciclos de especificação, desenvolvimento e validação efetivados a cada versão, permitindo sempre que

novas funcionalidades sejam parte da versão em atualização (PRESSMAN, 2011).

A Figura 4 dispõe representação do Modelo em Desenvolvimento incremental permitindo melhor entendimento de suas fases.

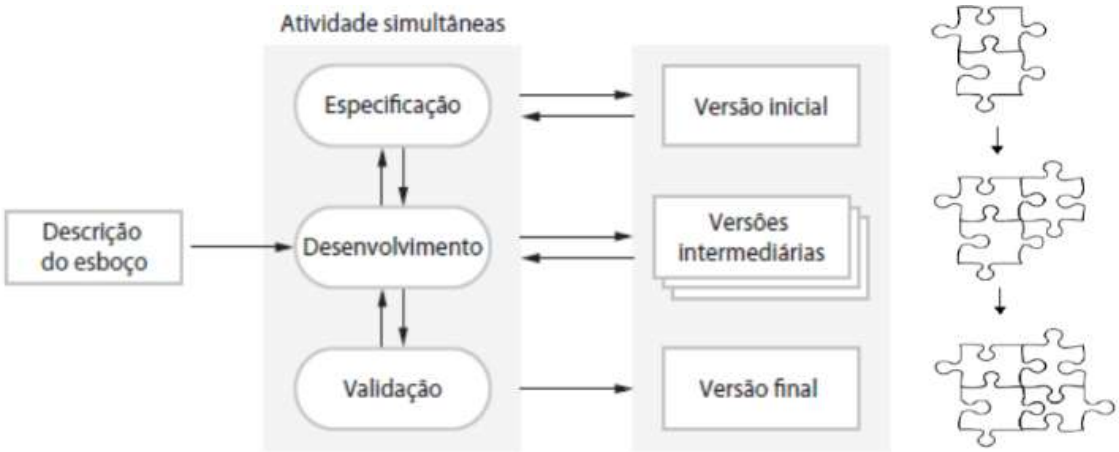

Figura: 4
Representação dos estágios do Modelo em Desenvolvimento Incremental

Na figura, observa-se por meio do diagrama iniciado na Descrição do esboço, que para a passagem de uma etapa a outra ocorrem interações, e somente após o seu pleno atendimento são efetivados os incrementos para as etapas seguintes.

Complementando, a disposição composta por peças de um quebra - cabeças demonstra que a cada etapa uma parte do produto vai sendo adicionada até que ocorra a sua conclusão.

Cabe destacar, que este modo de ação foi concebido em correção ao Modelo Tradicional em Cascata, no qual os ajustes somente se fariam efetivados ao final de todo o processo de desenvolvimento e apenas sobre os pontos de maior criticidade (WAZLAWICK, 2013).

Sua indicação de uso se faz nos casos em que se necessita privilegiar a liberação de alguma funcionalidade do sistema ou quando os profissionais envolvidos carecem de melhor distribuição nas atividades a serem desenvolvidas, permitindo que a gestão do tempo de execução seja mais efetiva (PRESSMAN, 2011; PRESSMAN; MAXIM, 2016).

1.2.5 Modelo em entrega Incremental

No Modelo em Entrega Incremental o cliente aponta quais as funcionalidades do sistema são de maior importância. Assim, posicionando-os como de maior prioridade em desenvolvimento, estes são logo incrementados favorecendo a efetivação de entrega e aptidão para operacionalização (SOMMERVILLE, 2011).

Como vantagem ao uso deste modelo tem-se, entre outras, a possibilidade de que as primeiras versões, antes do incremento final, sirvam de protótipo, favorecendo a experimentação do usuário/cliente e a troca de informações com a equipe de desenvolvimento (PRESSMAN, 2011; PRESSMAN; MAXIM, 2016).

A Figura 5 dispõe representação dos estágios do Modelo em Entrega Incremental permitindo seu melhor entendimento.

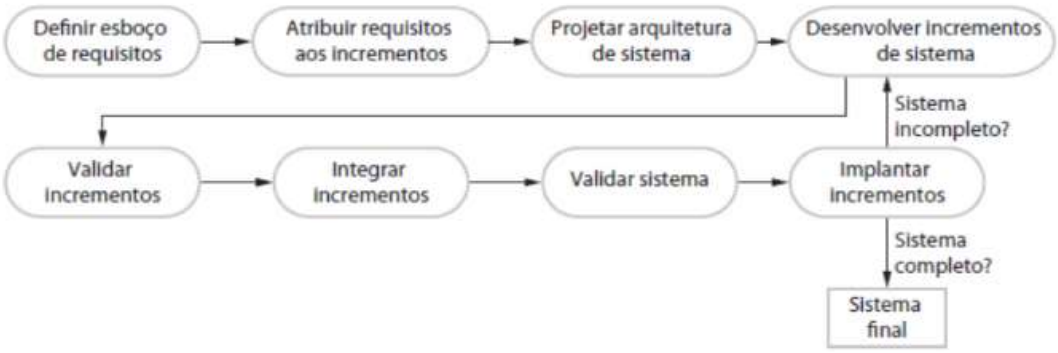

Figura: 5
Representação dos estágios do Modelo em Entrega Incremental

A figura demostra que garantir as finalização do sistema e a efetivação de seus incrementos são preocupações do modelo, garantindo que a possibilidade de especificação não se efetive apenas no início do projeto.

No entanto, vale destacar que o modelo de entrega Incremental não é indicado quando o sistema for de maior porte, o que requere a divisão de tarefas em equipes, e nem quando o sistema for desenvolvido para processos de maior criticidade e periculosidade (WAZLAWICK, 2013).

1.2.6 Modelo Orientada ao Reuso

O Modelo Orientado ao Reuso se aplica nos casos em que o desenvolvimento de sistema estiver focado na integração de módulos já existentes, restando identificar se os componentes passíveis de integração são aptos ou não ao recuso (SOMMERVILLE, 2011; PRESSMAN; MAXIM, 2016).

Basicamente, são orientados ao reuso os Serviços Web, as coleções de Objetos em Frameworks de Componentes e Sistemas Stand-Alone, ou seja, que funcionam individualmente e isoladamente, sem interação com outros (WAZLAWICK, 2013).

A Figura 6 dispõe representação dos estágios do Modelo Orientado ao Reuso permitindo seu melhor entendimento.

Figura: 6
Representação dos estágios do Modelo Orientado ao Reuso

Assim, diminuindo a quantitativo de software a ser desenvolvido, este modelo se aplica na redução de custos e do tempo de execução do projeto, contudo, a possibilidade de que os requisitos não se mantenham sintonizados com as necessidades dos usuários e com os módulos entregues no sistema é elevada, levando por vezes à necessidade de retrabalho e até novos

desenvolvimentos de software desde o processo inicial (PRESSMAN, 2011, HIRAMA, 2012; PRESSMAN; MAXIM, 2016; PETERS, 2001).

Assim, em se tratando da manutenção de software, ou seja, da necesidade de ajustes após a entrega do produto em si, cabe destacar os tipos de manutenção existentes, os quais seguem especificadas no Quadro 5.

Corretiva	Efetivada após a entrega do produto no intuito de corrigir um erro descoberto em garantia de pleno funcionamento do sistema.
Adaptativa	Efetivada após a entrega do produto visando estabelecer ajustes/alterações que garantam o uso do sistema.
Perfectiva	Efetivada após a entrega do produto no intuito de melhorar o desempenho e/ ou a manutenibilidade.
Preventiva	Efetivada após a entrega do produto para o reparo de falhas antes que o sistema se faça afetado.

Quadro: 5

Sendo a manutenção um requisito necessário e relacionado diretamente com a qualidade de software (manutenibilidade), cabe que o Processo de Software incorpore em seu ciclo de execução esta fase, garantindo que o sistema não se torne obsoleto pelo simples fato de sua desatualização.

2. ATIVIDADES RELACIONADAS AO PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

2.1 Especificação de software

A Especificação de Software tem como função compreender os serviços que se fazem requisitados para o sistema assim como identificar os aspectos que podem se posicionar como restritivos para o seu desenvolvimento (SOMMERVILLE, 2011).

Sua ação de fundamental importância pode se posicionar como marco de sucesso/insucesso para o Processo de Software e, justamente em face deste

aspecto, requer senso de responsabilidade e conhecimento para a boa efetivação dos processos desenvolvidos (WAZLAWICK, 2013; VAZQUEZ; SIMÕES, 2016).

Volta-se portanto a documentar os requisitos de especificação do si stema, permitindo que tanto usuários quanto desenvolvedores conheçam claramente e detalhadamente que aspectos deverão ser cobertos pela ferramenta computacional em desenvolvimento (PRESSMAN, 2011).

Sommerville (2011) e Pressman (2011) apresentam as atividades que fazem parte do processo de Especificação de Software e estas seguem apresentadas no Quadro 6.

Estudo de viabilidade	Dada a necessidade de se desenvolver um sistema, investiga-se a possibilidade de a ferramenta a ser proposta efetivamente atender as necessidades do usuário partindo do ambiente computacional de hardware e software existentes, da interação com os clientes/usuários, do orçamento disponível e do ambiente de negócio vigente na organização.
Elicitação e análise de requisitos	Partindo do entendimento de que a organização possui ou não um ambiente computacional prévio de hardware e de software, efetiva-se interações com os usuários /clientes a fim de identificar suas reais necessidades as quais impulsionam para o desenvolvimento de novos sistemas, permitindo inclusive a idealização de ferramentas por meio de modelos voltados à prototipagem.
Especificação de requisitos	Tendo estudado a viabilidade e efetivada a elicitação de requisitos, parte-se para o levantamento de informações efetivas e que devem ser registradas por meio dos requisitos de usuário (abstrações das necessidades efetivas do cliente e do usuário) e requisitos de sistema (descrição das funcionalidades a serem implementa das pelo profissional de TI).
Validação de requisitos	Constata se os requisitos especificados serão efetivamente atendidos na ferramenta a ser desenvolvida, permitindo a correção das especificações.

Quadro: 6

Cabe o destaque quanto os modos de levantamento de informações junto a clientes/usuários na fase de Estudo de Viabilidade, de modo que estes podem se dar por meio de entrevista, questionário, reuniões, observações, análises documentais (WAZLAWICK, 2013; PRESSMAN, 2011).

Quanto ao processo de Análise de Requisitos, naturalmente serão levantados Requisitos Funcionais e Não Funcionais. Em relação aos Requisitos Funcionais, tem-se que estes retratam as funcionalidades do software no intuito de que se planeje o treinamento dos usuários (exemplo: dados de cadastros de cliente, dados de cadastro de pedido); já os Requisitos Não Funcionais referem-se às estratégias utilizadas para que o sistema computacional esteja apto a resolver as problemáticas para as quais se fez desenvolvido (exemplo: efetivação de login e de autenticação de usuário) (WAZLAWICK, 2013; VAZQUEZ; SIMÕES, 2016; HIRAMA, 2012).

Isto, se bem definido auxiliará no pleno entendimento das necessidades do cliente/usuário, na manutenção futura do sistema, na redução do esforço desempenhado para o desenvolvimento, na diminuição dos custos e tempos de entrega e na validação final do sistema (WAZLAWICK, 2013; PETERS, 2001)

Em se tratando da Validação de Requisitos, é importante atentar para o papel do cliente/usuário neste processo, pois é justamente o seu feedback que dará ao desenvolvedor o entendimento quanto ao cumprimento dos requisitos funcionais e não funcionais levantados (PRESSMAN, 2011; VAZQUEZ; SIMÕES, 2016).

Contudo, cabe destacar que a Especificação de software não é uma ação que se deve efetivar apenas no início do Processo de software, sendo importante que durante todo o cumprimento das ações de desenvolvimento do software a especificação se faça presente (PRESSMAN, 2011).

2.2 Projeto e implementação de software

O projeto de Software descreve o software a ser implementado considerando a ideia elaborada para sua estrutura e funcionalidades, partindo da ação conjunta de uma equipe de projetistas que juntos estabelecem e revisam os elementos que deverão fazer parte do projeto em sua versão final (SOMMERVILLE, 2011).

No Projeto de um Software não se considera informações únicamente da ferramenta a ser desenvolvida, pois faz-se essencial considerar todo o macroambiente tecnológico em que ela se fará inserida, ou seja, a Plataforma de software a qual se faz composta pelo harware, sistema operacional, base de dados e outros (WAZLAWICK, 2013).

Daí se identifica a importância e relação desta fase com a de Especificação de Software: é justamente na etapa anterior que todas estas informações da Plataforma de software serão identificadas, de modo a garantir que a ferramenta a ser desenvolvida se faça apta a usufruir das funcionalidades do ambiente para o qual ela será implementada (PRESSMAN, 2011).

Pressman (2011) e Sommerville (2011) apresentam as atividades que fazem parte do Projeto de Software, e estas seguem apresentadas no Quadro 7.

Projeto de Arquitetura	Compreende a estruturação geral do sistema, considerando seus componentes, relacionamentos e distribuição destes.
Projeto de Interface	Compreende a definição de interfaces dos componentes e conexão entre estas interfaces.
Projeto de componente	Compreende o funcionamento ou as alterações que devem ser efetivadas para cada componente do sistema.
Projeto de Banco de Dados	Compreende o projeto de estrutura de dados, sua representação

Quadro: 7

Sendo consequência do projeto, a Implementação de Software deve ocorrer intercalado ao projeto, de modo a garantir que sempre possa ser efetivada a revisão das ações implementadas (SOMMERVILLE, 2011).

A experiência do projetista, sua capacidade de abstração, o uso ajustado de sua intuição e a aplicação de ferramentas de projeto reconhecidas e bem

desenvolvidas para esta finalidade devem representar papel importante na etapa de Projeto de Software, e é justamente tais fatores que serão de impacto na decodificação por meio da linguagem escolhida para o projeto (PRESSMAN, 2011).

Neste sentindo, o Quadro 8 apresenta aspectos que são fundamentais para todo e qualquer Projeto de Software.

Abstração	Solução identificada em atendimento ao problema relatado pelo cliente/usuário, podendo se apresentar sob a forma de procedimentos, algoritmos e código fonte.
Abstração de dados	Dados processados, agrupados, características e favoráveis a descrever um dado objetivo.
Modularidade	Divisão do software em partes, módulos, que podem interagir internamente e acoplar-se com outros externamente.
Procedimento de software	Especificações definidas para que cada módulo processe os seus próprios dados, podendo assim alimentar os demais.

Quadro: 8

Não havendo um padrão que obrigatoriamente deva ser seguido por todos os desenvolvedores, a equipe pode adotar uma abrodagem única, no entanto a preferência de por onde dar início ao trabalho de decodificação dependerá unicamente do profissional de desenvolvimento e de suas habilidades (WAZLAWICK, 2013).

2.3 Validação de software

A validação de software tem o papel de verificar se o software segue as especificações levantadas e se isto atende as necessidades apontadas pelo cliente/usuário (SOMMERVILLE, 2011).

A principal técnica utilizada para a validação é o Teste de Software, dando-se principalmente durante e após o processo de transformação do projeto em código, ou seja, da implementação do sistema (WAZLAWICK, 2013).

Contudo, relata-se que muitos erros de sistema a serem encontrados durante os testes podem se dar após a implementação completa, levando o retrabalho do desenvolvedor e ao atraso na entrega do produto (PRESSMAN, 2011). Pressman (2011) e Sommerville (2011) apresentam os estágios do Processo de Testes de software, e estes seguem apresentados no Quadro 9.

Testes de desenvolvimento	Os testes são efetivados em componentes isoladamente pelos próprios desenvolvedores.,
Testes de sistema	Estando os componentes integrados o sistema como um todo é testado no intuito de se encontrar erros e possibilidades de não atendimento aos requisitos.
Testes de aceitação	O cliente fornece dados para que se efetive testes simulados no sistema, podendo revelar problemas nos requisitos e no desempenho.

Quadro: 9
Estágios do Processo de Testes

Os testes de aceitação podem ser efetivados até que as falhas sejam completamente sanadas, neste caso, tem-se os testes alfa.

Contudo pode ocorrer casos em que o produto de software está finalizado e passa a ser utilizado por um grupo de usuários no intuito de que estes relatem suas experiências através dos testes beta, efetivados no intuito das devidas correções até que se torne apto para a comercialização/distribuição em definitivo (PRESSMAN, 2011).

2.4 Evolução de software

Os softwares envelhecem passando anteriormente por processo de nascimento, maturação e estabilidade, e por isso necessitam de atenção especial para que se mantenham funcionalmente em atividade, atendendo as demandas dos usuários (PRESSMAN, 2011).

Assim, no intuito de que, neste processo evolutivo o softwar e não venha a declinar e consequentemente “morrer”, faz-se primordial a implementação de ações de manutenção que favoreçam a constante evolução do sistema, permitindo assim que sejam efetivadas adequações voltadas à

interação entre software e hardware, garantindo que se mantenham em harmonia e plena produtividade (WAZLAWICK, 2013).

Tornar então o software apto para a evolução, tornando melhor o desenho de sua estrutura é um meio de garantir a longevidade do sistema, assim como documentar todo o projeto, desenvolvimento e manutenções permitindo que no futuro outras equipes de TI também possam garantir a operacionalização e produtividade do ambiente computacional já existente (PRESSMAN, 2011).

A Evolução do Software é composta por ações únicas que muito se adequam quando não há viabilidade no desenvolvimento de um software des de seu estágio inicial, tratando-se, portanto dos ajustes necessários para que o sistema se mantenha em pleno atendimento às necessidades de clientes/usuários mesmo havendo a mudança de requisitos de hardware na infraestrutura de tecnologia da organização (SOMMERVILLE, 2011; VAZQUEZ; SIMÕES, 2016).

Neste sentido, e servindo de embasamento para o desenvolvimento de modos diferenciados de se compreender o desenvolvimento de software a partir de metodologias ágeis, nos anos 70 foram lançadas as Leis de Lehman e estas seguem devidamente apontadas no Quadro 10.

Lei da Mudança contínua
A manutenção do sistema o adapta para que atenda às necessidades do usuário, requerendo portanto interação constante, mesmo que elas se alterem de modo crescente, sob pena de que a produtividade e satisfação sejam afetados consideravelmente ao longo do tempo.
Lei da complexidade Crescente
Sempre que um software sofre ajustes o seu nível de complexidade tende a aumentar, principalmente quando tais alterações não seguirem um planejamento estruturado de engenharia, podendo levar à necessidade de reestruturação do sistema caso alcance o nível de não mais atender às necessidades, produtividade e satisfação do usuário.
Lei da Autorregulação
Os esforços dispendidos a cada versão e no tamanho do sistema não tende a variar durante a vida do sistema, seguindo as diretrizes definidas pela organização solicitante da ferramenta.
Lei da Conservação da Estabilidade Organizacional ou Lei da Taxa Constante de Trabalho

Será contínua a necessidade de evolução do sistema, contudo isto não demandará aumento na equipe de tecnologia sob a pena desta se tornar improdutiva.
Lei da Conservação de Familiaridade
O incremento de ajustes na evolução de um software é constante a cada versão, sendo importante que a equipe de tecnologia envolvida se mantenha harmoniosa e alinhada aos objetivos que necessitam ser seguidos.
Lei do Crescimento Contínuo
Um sistema não deve ser ajustado apenas a nível de correção de erros, mas também na adição de novas funcionalidades, aumentando assim a satisfação do usuário.
Lei da Qualidade Decrescente ou Declinante
A evolução do sistema é parte de seu processo de uso, elevando a qualidade. Assim, a ausência de mudanças não é boa como se pensa, sendo importante que os ajustes aconteçam em prol da elevação da qualidade por meio do reajuste do sistema ao desenho dos processos.
Lei do sistema de Retorno ou de Feedback
Faz parte do processo evolutivo de software que usuários efetivem retornos avaliativos tanto positivos quanto negativos, pois é justamente estas ações que favorecem os ajustes e a evolução do sistema evitando sua “morte” prematura.

Quadro: 10
Leis de Lehman acerca da evolução de software

Assim, foi importante observar que a Evolução de software incorpora em si o entendimento da necessidade de manutençãi, pois sem esta ação se faz possível os ajustes ao longo da vida útil do sistema.

Contundo, as constantes alterações em software levam à possibilidade de que o projeto de desenvolvimento nunca consiga alcançar uma finalização, sem que ao menos as versões sejam efetivamente finalizadas. A este tipo de contexto denominna-se de Beta Eterno, onde, apesar de se percorrer todas as etapas tradicionais de desenvolvimento, o que se faz liberado ao usuário é uma versão beta, a qual ainda passará por uma inifinidade de adequações que em tese levariam à finalização de sua elaboração (ALÃO, 2018).

Diante deste modo de pensar, a possibilidade de dispor o software a uma quantidade grande de usuários antes de sua finalização, funcionaria como meio de aceleração no desenvolvimento, contudo, conforme destaca Alão (2018), não é o que acontece, pois, a exemplo de gigantes da tecnologia como o Google,

tem-se a prática como meio de distorcer o entendimento do usuário de que sendo uma versão beta, as falhas são perdoáveis.

Isto coloca a necessidade de reflexões em relação aos processos praticados em um mundo de versões beta, principalmente se considerando as Leis de Lehman em relação à evolução de software.

3. ESTIMATIVA DE SOFTWARE

3.1 Conceito e importância da estimativa de software

O termo estimativa, conforme o Dicio (2023), refere-se ao cálculo efetivado no intuito de se avaliar algo/alguém, tendo por base evidências que permitam melhor conhecer e quantificar estatísticas acerca do ser/objetivo avaliado.

Aliado a este entendimento, Wazlawick (2013) destaca que, dificilmente uma atividade profissional, seja ela qualquer, não possua suas próprias maneiras de estimar o esforço aplicado para obtenção de um dado resultado, favorecendo assim, para que no âmbito gerencial se alcance um maior controle das ações desempenhadas assim como da qualidade alcançada em relação ao elemento produzido.

Neste intento de estimar, tem-se o entendimento de medição, que segundo Vazquez et al. (2018) e Wazlawick (2013), refere-se ao processo direto de atribuir números no intuito de quantificação das características atribuídas a um determinado elemento, necessitando-se assim, identificar que atributos são estes de tamanha relevância a ponto de despertar o interesse para sua mensuração.

Por conseguinte, a métrica se refere à quantificação indireta, e nela tem-se o cálculo aplicado sobre a medição, destacando qual o critério que se fará avaliado, ou seja, a característica que será medida e que já fora anteriormente identificada como de destaque/relevância (WAZLAWICK, 2013).

Assim, o resultado alcançado a partir da aplicação de uma medição tem o intuito de permitir que as pessoas envolvidas no des envolvimento de um dado

projeto compreendam como os elementos envolvidos podem afetar cliente e usuário, viabilizando o conhecimento e a correção dos problemas rapidamente e sem que consigam acarretar maiores impactos (VAZQUEZ ET AL., 2018).

Diante destes entendimentos, Sommerville (2011) destaca que, em se tratando da medição de software, tem-se a possibilidade de, por meio dela e dos elementos característicos do processo de mensuração, identificar questões referentes à qualidade do software desenvolvido, documentação e/ou qualidade dos processos que permeiam as tarefas desempenhadas.

3.2 Métricas de Software

Em se tratando especificamente de Métricas de Software, vale destacar que o termo se refere ao processo de medir algo ao relação a um dado atributo que seja de relevância para o software (PRESSMAN; MEXIM, 2016)

Neste contexto a categorização de medidas para a construção de software pode ser fazer atribuída em nível de medidas de produto, projeto e recurso, como segue melhor identificado no quadro abaixo.

Medidas de Produto	Refere-se a característocas como tamanho, estrutura e qualidade.
Medidas de Projeto	Refere-se aos desempenho de um software podendo ser calculado por porcentagem.
Medidas de Recursos	Refere-se aos recursos e materiais aplicados no desenvolvimento do projeto de software.
Medidas de Processo	Refere-se à coleta de informações que permitem identificar os pontos fortes, fracos, deficiências, e avaliação após implementação/mudança em software.

Quadro: 11
Medidas de Software

É justamente a partir destas medidas que são especificadas as métricas a serem atribuídas para a medição de software, de modo a garantir que dúvidas a respeito do projeto de software possam efetivamente ser respondidas.

Por conseguinte, as métricas de software podem ser categorizadas conforme se verifica no Quadro abaixo.

Méttricas de controle ou Métricas de Projeto	Embasa o gerenciamento de processos de software, especificando meios de avaliação do seu andamento, riscos, problemáticas, fluxo de tarefas e habilidades da equipe, sendo coletadas durante toda a efetivação do projeto a fim de permitir o estabelecimento de indicadores de melhoria a longo prazo.
Métricas de Previsão ou Métricas de Produto	Embasam a previsão e mediação dos atributos internos de um software, ajudando a estimar o esforço dispendido para o desenvolvimento e alterações do sistema em face de seu grau de complexidade, seu tamanho, riscos e tecnologia aplicada.

Quadro: 12
Métricas de Software

Demonstra-se assim que as métricas ultrapassam a visão da simples efetivação de controle, permitindo que variáveis importantes sejam consideradas no projeto de desenvolvimento do software (PRESSMAN; MAXIM, 2016).

Reconhece-se, por tanto, o papel das métricas em permitir que o desenvolvimento de software seja constantemente aperfeiçoado, uma vez que as informações alcançados embasam a tomada de decisão da gestão e da equipe desenvolvedora, favorecendo o processo de planejamento (PRESSMAN, 2011; WAZLAWICK, 2013).

A estimativa de software não se efetiva de modo aleatório, podendo fazer uso de técnicas já estabelecidas que passaram por todo um processo de análise e testes até se tornarem referência, garantindo que o planejamento forneça efetivamente a direção que deve se fazer ser seguida pelos envolvidos (PRESSMAN; MAXIM, 2016).

Neste âmbito, segue a apresentação de ferramentas passíveis de utilização visando as efetivação de estimativas de software: a Análise de Pontos de Função e o Planning Poker.

3.2.1 Análise por ponto de função

A Análise por Ponto de Função (APF) ou Function Point Analysis (FPA) configura-se como técnica voltada à medição das funcionalidades,

dimensionamento e complexidade de um software por meio de processos aplicados ao tamanho do projeto, contundo, tendo por parâmetro o ponto de visão do usuário (PRESSMAN; MAXIM, 2016; VAZQUEZ et al., 2018).

Como os métodos de medição identificam e quantificam as qualidades existentes em um produto, em se tratando de Pontos de Função, tem-se um processo que mede os requisitos funcionais atribuíveis ao usuário em um software (histórias de usuário), ou seja, os requisitos de negócio, sem estabelecer atenção ao as pecto tecnológico, de plataforma e/ou linguagem, focando exclusivamente na ação direta da ferramenta desenvolvida (VAZQUEZ et al., 2018).

> Como já especificado, considera-se como requisitos funcionais aqueles que expressam as ações do software ao ser aplicado na atividade do usuário, de modo que, caracterizam elementos aplicados ao processo de negócios da organização, podendo se fazer relacionados à transferência e armazenagem de da dos (WAZLAWICK , 2013).

Deste modo, aos Pontos de Função, sendo os requisitos que passarão por medição, efetiva-se a atribuição de valores numéricos, os quais processados posteriormente, permitem identificar o esforço dispendido durante o desenvolvimento do software, pois em se tratando deste aspecto, amplia-se o entendimento considerando esforço, prazo, custo e demais parâmetros, o que pode em projetos de software respaldar as conclusões acerca do seu tamanho e complexidade (WAZLAWICK, 2013; VAZQUEZ et al., 2018).

Nesta base, destaca-se os seguintes elementos dispostos na figura abaixo como objetivo atribuíveis à Análise por Pontos de Função.

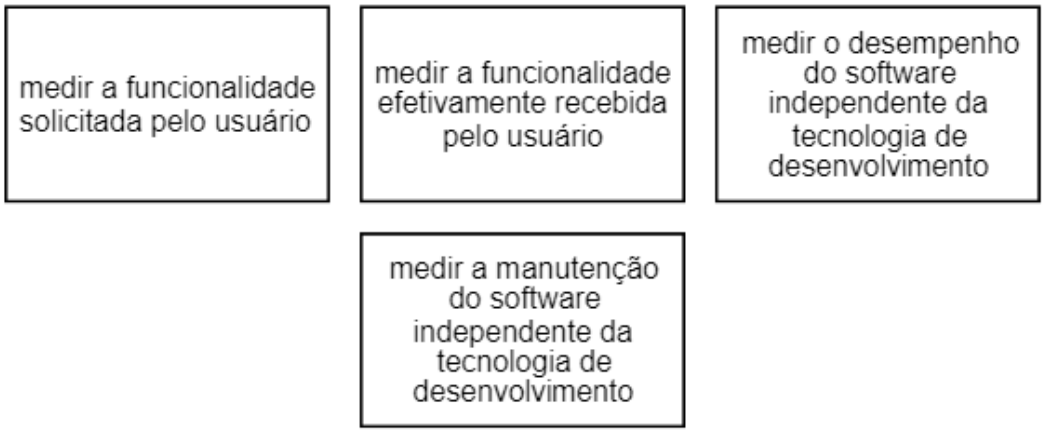

Figura: 7

Efetivando-se a Análise por Pontos de Função em atributos do sistema, aos quais compreende-se que sejam os requisitos, o resultado alcançado pelo processo também pode se voltar a intenções que não estejam elencadas na Figura 7, cabendo ainda para outros intentos como os apontados por meio da Figura 8.

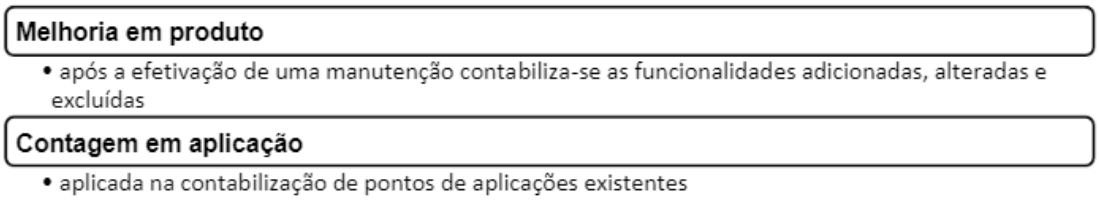

Figura: 8

Para tal, o processo que envolve a Análise por Pontos de Função necessita seguir os parâmetros dispostos pela figura 9:

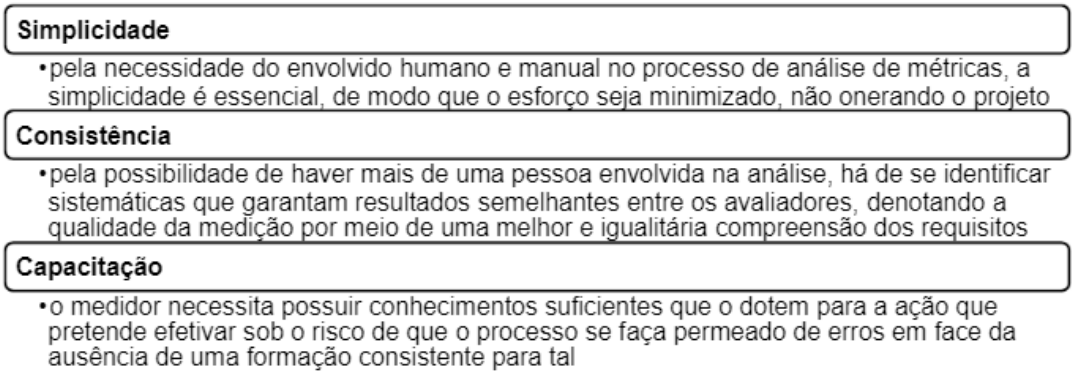

Figura: 9
Parâmetros ideais para a Análise por Pontos de Função

A contagem de Pontos de Função, visando sua posterior análise, efetiva-se de modo simplificado por meio da sistemática dispostas na figura 10.

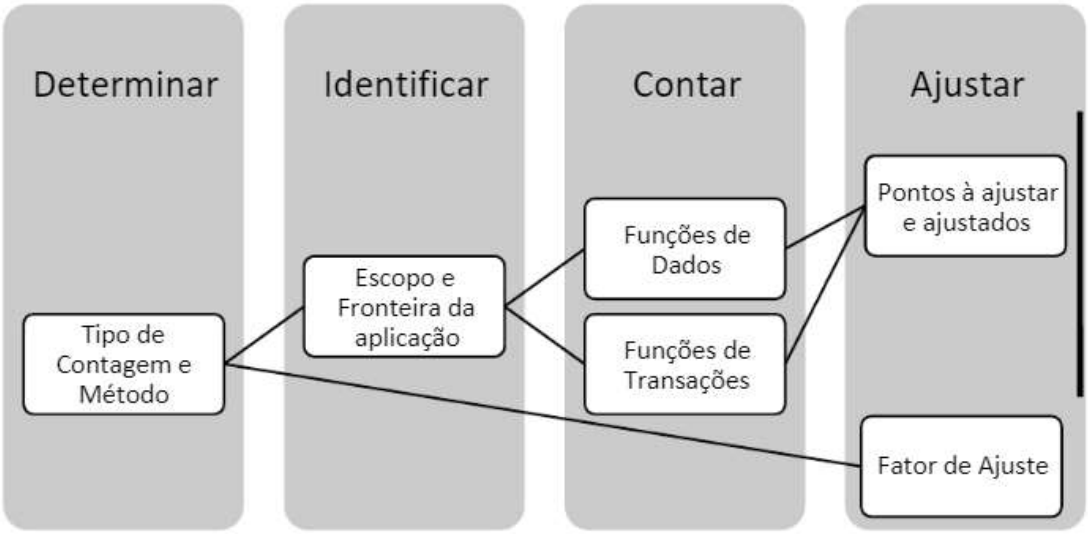

Figura: 10
Sistemática simplificada de contagem de Pontos de Função

Neste contexto, em se tratando dos elementos destacados na sistemática simplificada presente no meio da figura 11, tem-se de modo mais abrangente um detalhamento de ações apresentadas através do quadro 13.

AÇÕES	PROCESSOS	ESPECIFICAÇÃO
Determinar	Tipo de Contagem	• Projeto • Melhoria • Aplicação
	Método	• Estimado • Detalhado
Identificar	Escopo e Fronteira da Aplicação	• Limite entre sistema e usuário • Limite entre sistema e outras aplicações • Visão do Usuário • Função de Negócio • Independência de Tecnologia • Manutenção
Contar	Funções de Dados	• Arquivos Lógicos Internos (ALI) • Arquivos de Interface Externa (AIE)
	Funções de Transação	• Entrada Externa (EE) • Consulta Externa (CE) • Saída Externa (SE)
Ajustar	Fator de Ajuste	• Atualização Online • Complexidade de Processamento • Comunicação de Dados • Configuração Altamente Utilizada

	Pontos a ajustar	• Eficiência de Usuário Final • Entrada de Dados Online • Facilidade de Instalação • Facilidade de Mudanças • Facilidade de Operação • Múltiplas Localidades • Performance • Processamento Distribuído • Reutilização • Taxa de Transações
	Pontos ajustados	

Quadro: 13

Considerando a figura 10 e o Quadro 13, e compreendendo que a dimensão dos requisitos funcionais do usuário recebe a denominação de tamanho funcional e que uma aplicação seja um conjunto composto de dados e procedimentos que sofreram automatização fornecendo suporte ao processo de negócio de acordo com a ação do usuário (VAZQUEZ et al., 2018) os Tipos de Contagem passíveis de efetivação na Análise por Pontos de Função, seguem explicitados por meio da Figura 11.

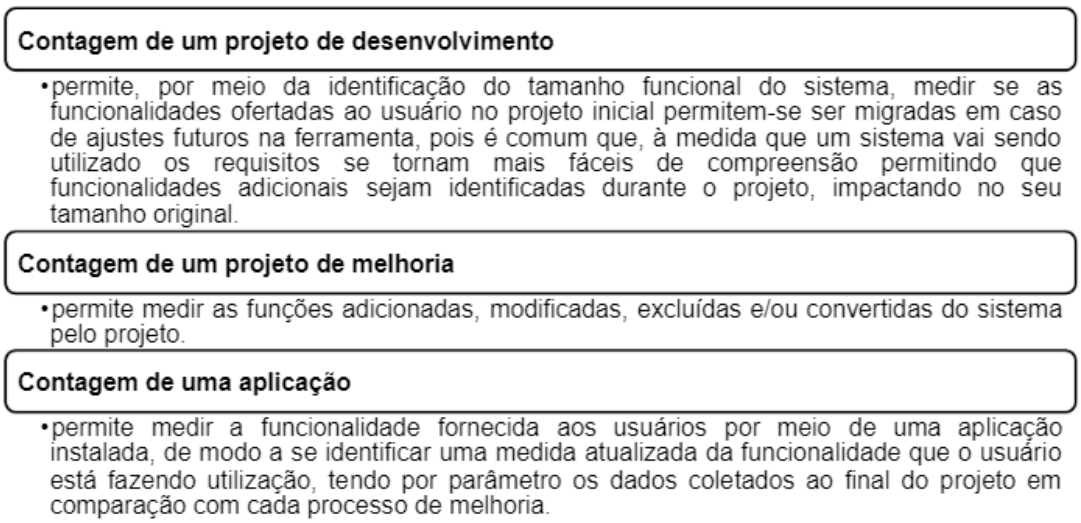

Figura: 11
Tipos de Contagem em Análise por Pontos de Função

Em se tratando do aspecto referente à Fronteira da Aplicação, tem-se a identificação dos limites entre o software e o mundo em que ele se faz inserido, seguindo os critérios de ponto de vista do usuário, a separação de funções atribuídas aos negócios, e, em caso de projetos de melhoria, os aspectos já definidos devem ser reanalisados com intuito de verificar se existe a

necessidade de adaptação (VAZQUEZ et al., 2018). Neste sentido, tem-se como aspectos facilitadores para a identificação das fronteiras de uma aplicação estes que se fazem citados na Figura 12.

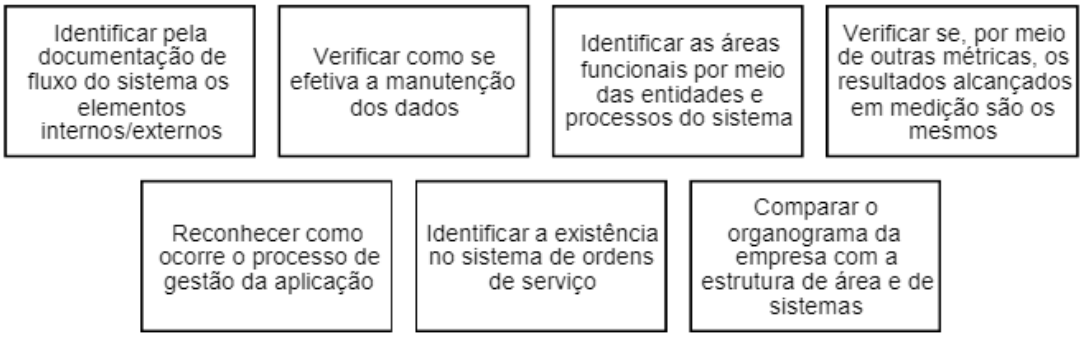

Figura: 12
Aspectos que facilitam a identificação das Fronteiras de uma aplicação

Referindo-se ao Escopo, tem-se a identificação de quais as funcionalidades a serem contadas por meio da sistemática escolhida, podendo incorporar todas as existentes, apenas aquelas que são utilizadas pelo usuário, ou ainda algumas em específico (VAZQUEZ et al., 2018).

Tem-se, deste modo, a identificação de quantos sistemas serão quantificados, definindo claramente onde começam e terminam (WAZLAWICK, 2013).

Por conseguinte, define-se o Tipo de Método, podendo, conforme Pressman e Maxim (2016) ser:

a) Estimado: Calcula os Pontos de Função de acordo com regras internacionalmente definidas no intuito de estimar as Funções de Dados (preferencialmente de complexidade baixa) e as Funções de Transação (preferencialmente de complexidade média);

b) Detalhado: Efetiva a quantificação dos Pontos de Função com regras internacionalmente definidas no intuito de estimar as Funções de Dados e de Transação de acordo com as especificidades do caso.

Em relação à contagem/medição das Funções de Dados e de Transação, identificar a natureza dos dados configura-se como o primeiro passo do processo, pois é justamente a partir dele que se reconhece as reais necessidades do usuário em relação ao negócio, assim como as funcionalidades previstas pelos usuários (VAZQUEZ et al., 2018; PRESSMAN; MAXIM, 2016; WAZLAWICK, 2013), sendo tais ações melhor especificadas por meio do Quadro 14.

DADOS	FUNÇÕES	ARQUIVOS	
Estáticos dados estruturados na forma de arquivos internos/externos	**Funções de Dados**	**Arquivo Lógico Interno (ALI)**	Grupo lógico de dados ou informações reconhecidos pelo usuário e que servem para efetivações de controle
		Arquivo de Interface Externa (AIE)	Grupo lógico de dados ou informações reconhecidos pelo usuário e que se fazem mantidos fora da aplicação
Dinâmicos dados de transação na forma de entradas, saídas e consultas	**Funções de Transação**	**Entradas Externas (EE)**	Dados informados pelo usuário que alimentam o sistema alterando seu estado interno
		Saídas Externas (SE)	Dados que saem sob requisição ou não de cliente/sistema, apresentando quantificações em relação a algum parâmetro previamente definido
		Consultas Externas (CE)	Dados que saem sob requisição ou não de cliente/sistema, do mesmo modo em que se fazem armazenados

Quadro: 14
Especificações acerca de Funções de Dados e Funções de Transação

Por conseguinte, após a identificação das funções a serem contadas, é requerida a identificação de seu fator de complexidade, devendo este seguir os parâmetros dispostos por meio do quadro 15.

TIPO DE FUNÇÃO	COMPLEXIDADE			PESOS
	Baixa	Média	Alta	
Entradas Externas (EE)	3	4	6	Tipos de Dados: <5 / 5-15 / >15 — Arquivos: <2 (Baixa, Baixa, Média); 2 (Baixa, Média, Alta); >2 (Média, Alta, Alta)
Saídas Externas (SE)	4	5	7	Tipos de Dados: <6 / 6-19 / >19 — Arquivos: < 2 (Baixa, Baixa, Média); 2-3 (Baixa, Média, Alta); > 3 (Média, Alta, Alta)
Consultas Externas (CE)	3	4	6	
Arquivo Lógico Interno (ALI)	7	10	15	Tipos de Dados: <20 / 20-50 / >50 — Arquivos: <1 (Baixa, Baixa, Média); 2-5 (Baixa, Média, Alta); >5 (Média, Alta, Alta)
Arquivo de Interface Externa (AIE)	5	7	10	

Quadro: 15
Tipos de Função conforme fatores de complexidade

Conforme os parâmetros apresentados no Quadro 15 referentes aos fatores de complexidade concernentes às funções de dado e transação, os pontos alcançados devem ser somados a fim de que se identifique o Tamanho Funcional da aplicação (WAZLAWICK,2013).

Isto se efetiva identificando-se os tipos de registro da Função de Dados e a quantidade de campos da Função de Transação, de modo que com a atribuição dos pesos de complexidade apresentados tem-se a quantificação dos Pontos de Função a Ajustar (PRESSMAN; MAXIM, 2016).

3.3 Planning Poker

O Planning Poker é uma técnica aplicável também à estimativa de complexidade, considerando o tamanho de projetos que incorporam metodologias ágeis e as histórias de usuário como elementos base para o levantamento de requisitos (COHN, 2005; GOMES, 2014).

Contudo, sua aplicação em conjunto com a Metodologia Scrum, apesar de bastante afinada, não se faz obrigatória, dependendo de definições

estabelecidas no planejamento do projeto, os quais se efetivam por meio da gestão (GOMES, 2014; SABBAGH, 2022).

Se aplica muito bem a equipes ágeis por seu dinamismo, requerendo apenas dedicação de tempo para execução, o que se faz compensado pelo retorno alcançado no processo (AUDY, 2022).

O Planning Poker age no sentido de pertencimento dos indivíduos envolvidos, no aprendizado coletivo e contínuo, na importância dos variados tipos de conhecimento e especialidades de pessoas que perfazem o corpo do time, e no papel do consenso para se alcançar o melhor resultado possível (AUDY, 2022).

Por meio da aplicação e do uso de um jogo de cartas, os indivíduos que perfazem a equipe de desenvolvimento posicionam-se, considerando a sua visão de complexidade em relação ao sistema, tendo por parâmetro os fatores tempo e esforço, os quais servem como elementos-base para que se alcance um consenso a respeito do assunto em voga (COHN, 2005).

4 REFERÊNCIAS:

ALÃO, Rui. Estratégias de design para contextos complexos. In: CONGRESSO PESQUISA & DESENVOLVIMENTO EM DESIGN, 13 ., Joinville, 5-8 nov. 2018. Anais [...]. Santa Catarina: Univille, 2018.

AUDY, Jorge. Scru m 360. Um guia completo e prático de agilidade . São Paulo: Casa do Código , 2022.

COHN, M. Agile Estimating and Planning. [s. l.]: Prentice Hall, 2005.

DICIO. Dicionário Online de Português. Estimativa. 2023.

GOMES, André Farias. Agile: desenvolvimento de software com entregas frequentes e foco no valor de negócio. São Paulo: Casa do Código, 2021.

HIRAMA, Kechi. Engenharia de Software: qualidade e produtividade com tecnologia. Rio de Janeiro : Elsevier, 2012.

OLIVEIRA, Bruno Souza de. Métodos Ágeis e Gestão de Serviços de TI. Rio de Janeiro: Brasport, 2018.

PETERS, James F., Engenharia de Software: Teoria e Prática. Rio de Janeiro: Campus, 2001.

PRESSMAN, Roger S. Engenharia de Software. 7. ed. São Paulo: Pears on Prentice Hall, 2011.

PRESSMAN, Roger S.; MAXIM, Bruce R. Engenharia de Software: uma abordagem profissional. 8 . ed. Porto Alegre: AMGH , 2016.

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pears on, 2011.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira. Engenha ria de Requisitos: software orientado ao negócio. São Paulo: Brasport, 2016.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. Análise de Pontos de Função: medição, estimativa e gerencia mento de projetos de software. 13 e d. São Paulo: Érica:Sarai va, 2018.

WAZLAWICK, R. S. Engenharia de software: conceitos e práticas. Rio de Janeiro: Elsiever, 2013.