SELINUS UNIVERSITY
OF SCIENCES AND LITERATURE

Simples Sequencia Para Controlar e Organizar o Desenvolvimento de Rotinas Administrativas com Engenharia de Software: Proposta de Implementação no Setor de Registro Hospitalar

MIGUEL MARCELO NASCIMENTO FRANCO

2022

Por honra e merecimento dedico este trabalho aos meus pais e meus irmãos a minha esposa e amigos que sempre me incentivaram.

AGRADECIMENTOS

Por honra e merecimento agradeço a instituição pela a oportunidade e aos meus orientadores e colegas por me ajudarem a desenvolver este trabalho.

"Vencer é o que importa. O resto é a consequência."
(Ayrton Senna)

RESUMO

Haverá uma breve explicação sobre o tema, contudo será nesse trabalho proposto um novo propósito de controle organizacional para o desenvolvimento de rotinas do software ou em outras palavras no que desrespeito ao sistema legado da empresa, a simples seqüência. Certamente, com base nos padrões de engenharia de software mais pertinentes e relevantes, pretende-se, contudo esclarecer e mostrar as mais diversas opiniões de autores e suas grandes obras da engenharia de software, seguidos de resumo bibliográfico, opinião formada para assim talvez firmar um novo propósito organizacional, cujo especulasse a idéia na qual será voltada principalmente para estruturar o código do software e organizá-los dentro dos padrões de desenvolvimento da engenharia de software. Contudo, o objetivo geral fundamenta-se em mostrar possíveis razões para acreditar que há possibilidade de sucesso nesse propósito organizacional. Portanto, para tal feito o foco será na pesquisa exploratória, juntamente com analise documental para obtenção dos dados, seguido da abordagem qualitativa para efeitos de analise e interpretação de dados. Ao chegamos ao final do trabalho há uma intenção de se ter fortes argumentos sustentados pelas as diversidades das citações e obras empregadas no trabalho do seu inicio ao fim.

Palavras-chave: Desenvolvimento. Software. Controle. Organização.

ABSTRACT

There will be a brief explanation on the subject, however, a new purpose of organizational control for the development of software routines will be proposed in this work, or in other words, in respect of the company's legacy system, the simple sequence. Certainly, based on the most pertinent and relevant software engineering standards, it is intended, however, to clarify and show the most diverse opinions of authors and their great works of software engineering, followed by a bibliographic summary, an opinion formed to perhaps establish a new organizational purpose, which would speculate the idea in which it will be mainly aimed at structuring the software code and organizing them within the software engineering development patterns. However, the overall objective is based on showing possible reasons for believing that there is a possibility of success in this organizational purpose. Therefore, for this purpose, the focus will be on exploratory research, together with document analysis to obtain the data, followed by a qualitative approach for the purposes of data analysis and interpretation. When we reach the end of the work, there is an intention to have strong arguments supported by the diversity of citations and works used in the work from beginning to end.

Keywords: Development. Software. Control. Organization.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de dados
CASE	Engenharia de software auxiliada por computador
ER	entidade relacionamento
HTTP	Protocolo de Transferência de Hipertexto
PDSS	Propósito da simples sequência
PLSQL	Linguagem procedural / linguagem de consulta estruturada
RES	Engenharia de requisitos do software
SGBD	Sistema gerenciamento de banco de dados
SLD	Sistemas legados distribuído
SSDLC	Ciclo de vida de desenvolvimento de sistemas e software
URL	Localizador Uniforme de Recursos

LISTA DE SÍMBOLOS

;	Ponto-e-vírgula
:	Dois Pontos
.	Ponto
@	Arroba
/	Barra
//	Barra Barra
&	E comercial

SUMÁRIO

1 **INTRODUÇÃO**

A padronização e melhoria de processos, bens e serviços se dá através da participação e do comprometimento de todos os colaboradores. A adoção de um sistema de gestão implica, na maioria das vezes, a padronização dos métodos e práticas de uma organização. Isso é importante pois permite que a análise crítica e a conseqüente melhoria dos procedimentos e métodos da empresa, possibilitando uma perspectiva concreta do que analisar e melhorar (MARSHALL JUNIOR et al., 2008).

Segundo (PALADINI, 2010), a integração das ações de suporte ao modelo gerencial confere à Gestão da Qualidade uma característica que considera relevante qualquer recurso, atividade ou função da empresa. Se esses elementos compõem a organização é porque possuem capacidade de contribuir para manter ou aumentar a adequação do produto ou do serviço ao uso.

A Gestão da Qualidade envolve ferramentas simples ou, eventualmente, mais elaboradas, destinadas a dar forma a suas ações. Essas ferramentas podem estar relacionadas à definição do melhor modo de atendimento aos clientes, à redução de custos ou ao modo de envolver funcionários em processos de análises de problemas para definir suas possíveis soluções. Como melhoria contínua, a Gestão de Qualidade abrange estratégias que visam a definir a melhor maneira de executar ações produtivas, partindo de situações existentes, procurando-se sempre melhorá-las (PALADINI, 2010).

Acredita-se que o desenvolvimento do código fonte dos softwares de diversas organizações está sendo feito de forma errada nesse contexto as empresas possuem softwares com códigos desestruturados, porém, nada impede que sejam estruturados de maneira correta, esses softwares também conhecidos como “sistema legado”, que podem ser muito antigos. Será nesse sentido apresentado um novo propósito que deve ficar alinhado com os padrões de desenvolvimento da engenharia de software que são: desenvolvimento, organização e centralização do código fonte da aplicação. O que se sabe até hoje é que os sistemas legados podem causar preocupação no quesito melhoria e adequação de suas rotinas, quem sabe até a sua evolução se torne complicada e demorada.

> Os sistemas de software legado foram desenvolvidos décadas atrás e tem sido continuamente modificado para se adequar ás mudanças dos requisitos de negócios e a plataforma computacional. A proliferação de tais sistemas está causando dores de cabeça para grandes organizações que os

> consideram dispendiosos de manter e arriscados de evoluir (PRESSMAN; MAXIM, 2016, p. 8).

Teremos um único objetivo que é responder a essa pergunta, como a engenharia de software pode melhorar o desenvolvimento de rotinas de processos e projetos do software, nesse novo propósito organizacional? Justifica-se que o estudo sobre o tema se tornará relevante, importante e contribuirá para pesquisas futuras relacionadas com temas pertinentes dessa área, contudo auxiliará o setor de desenvolvimento das organizações. O problema este, acredita-se na desorganização do código do sistema legado, portanto torna-se característica fundamental de nosso referencial teórico. A pesquisa partiu da hipótese de uma possível desorganização da estrutura interna do software e escassez do tema proposto que pode estar ligada diretamente a falta de conhecimento do tema pelos os profissionais que atuam diretamente ou indiretamente na melhoria das rotinas do sistema legado da organização. Então nasceu o tema em questão e a seguinte metodologia centrada em um propósito garantindo assim chega-se a um possível e determinado fim, que estar ligada as pesquisas bibliográficas, citações, coleta de dados, questionários, analise dos dados, procedimentos e resultados.

1.1 TEMA E ÁREA DE PESQUISA DO TRABALHO

Pode ser que este tema que resumidamente se refere a pergunta como a engenharia de software pode melhorar o desenvolvimento de rotinas de processos e projetos do software, nesse novo propósito organizacional? nascera do interesse relacionado ao estudo da engenharia de software que poderá ser discutido futuramente pelo o mundo ou até globalmente de acordo com o crescimento desse assunto que será discutido nesse trabalho onde o estudo poderá mostrar algo de concreto sobre o desenvolvimento de rotinas e distribuição do software. Segundo (SOMMERVILLE, 2011, p. 11) Conforme o avanço da internet “um software é altamente distribuído, às vezes pelo mundo todo. As aplicações corporativas não são programadas do zero; de fato, elas envolvem reuso extensivo de componentes e programas”, contudo, tais rotinas existentes nas aplicações das organizações corporativas não são desenvolvidas do zero. Portanto, o estudo apresenta base para o campo de pesquisa e referencial citado. Contudo, a uma relevância para o campo de estudo.

1.2 PROBLEMA DE PESQUISA DO TRABALHO

Havia uma duvida e com estar, se fez nascer esta pesquisa, a duvida partiu

da seguinte pergunta: como a engenharia de software pode melhorar o desenvolvimento de rotinas de processos e projetos do software, nesse novo propósito organizacional? No entanto arrastaremos até o final desse trabalho esse problema de pesquisa, tentaremos tirá-la utilizando para melhor compreensão do estudo, os procedimentos metodológicos onde se caracterizam por uma revisão bibliográfica, de cunho exploratório descritivo. Segundo (GIL, 2017, p. 43) pesquisa exploratória tem como finalidade proporcionar maiores informações sobre o assunto que se vai investigar; facilitar a delimitação do tema da pesquisa; orientar a fixação dos objetivos, descobrirem um novo enfoque para o assunto. Ainda, em relação à pesquisa descritiva, relata o citado autor, que esta “tem como objetivo primordial à descrição das características de determinada população ou fenômeno ou, então, o estabelecimento de relações entre as variáveis”. As pesquisas descritivas são, juntamente com as exploratórias, as que habitualmente são realizadas pelos pesquisadores sociais, preocupados com a atuação prática. Assim, levanta-se o seguinte problema de pesquisa: como a engenharia de software pode melhorar o desenvolvimento de rotinas, processos e projetos, nesse novo propósito organizacional?

1.3 JUSTIFICATIVA E MOTIVAÇÃO PARA RESOLVER

Poderemos justificar o desenvolvimento dessa pesquisa na medida em que se reconhece que o grande desafio da organização é fazer seus líderes perceberem a importância de melhorar e aperfeiçoar seu software. Que além da necessidade de implementar melhorias no sistema, há a necessidade de manter os profissionais da organização satisfeitos em realizar a manutenção do sistema legado da empresa. Portanto o estímulo para a realização deste estudo surgiu da motivação e amor à profissão de analista de sistemas, bem como do desejo de compartilhar possíveis descobertas que permitam compreender melhor o tema citado, observando as diferenças de comportamento de cada processo.

O projeto será implantado numa empresa do ramo hospitalar, o qual não possuía o acompanhamento dos custos do negócio, planejamento e procedimento das atividades a serem executadas, gerando aumento nas despesas, desprogramação dos setores, desmotivação e perda de dados cadastrais e lentidão nos processos do setor operacional de serviços.

Espera-se que, com o presente trabalho, seja possível apresentar conceitos e implantar o processo de padronização e o gerenciamento da rotina de trabalho em

cada departamento/ setor administrativo da empresa, através do controle sistemático e da melhoria contínua de cada atividade. Com metas estipuladas para cada uma delas, pretende-se buscar a melhoria contínua dos procedimentos através de ferramentas administrativas adequadas, obtendo a excelência nos serviços prestados e a satisfação dos clientes internos e externos.

Com o objetivo empresarial estabelecido e as metas definidas, permite-se uma melhor distribuição de tarefas e melhor acompanhamento dos resultados parciais, permitindo a cada unidade básica desenvolver seu plano de ação, envolvendo os aspectos técnicos, administrativos e operacionais relacionados (LOBATO et al., 2009).

Segundo (BERSSANETI; BOUER, 2013) o gerenciamento da rotina é um processo permanente e contínuo em base diária, acontecendo, portanto, durante cada micro processo de um departamento. Porem, os requisitos para a aplicação dessa ferramenta é a existência de micro processos repetitivos, definidos operacionalmente (padronização), interesse em aperfeiçoar os micros processos, vontade em dar um significado à Qualidade também fora do âmbito da produção, e o desejo de querer trabalhar de forma sistemática sobre uma base de dados confiáveis para identificar e agir prontamente sobre os gargalos do micro processo. Nota-se (BERSSANETI; BOUER, 2013), o gerenciamento de rotina, também conhecido como gerenciamento do cotidiano, entretanto, é indicado em situações em que uma área operacional de uma empresa constata que o seu desempenho não estar satisfatório. Porém, o reconhecimento dessas situações de desempenho abaixo do esperado, a caracterização dos problemas a ele relacionados e a resolução desses problemas são pontos fundamentais no gerenciamento da rotina.

Dessa forma, percebe-se a importância das definições, controles e acompanhamento dos processos realizados por cada setor da empresa, garantindo o sucesso dos objetivos da organização.

Segundo (FRANCO, 2020, p. 11). em sua obra, a “escassez de estudos relacionados com o tema proposto e o interesse em compreender os desafios enfrentados pelos profissionais que atuam nas organizações, atuantes no processo de desenvolvimento de rotinas dos softwares, sistemas legados empresariais, assim como, a busca por maiores e melhores oportunidades dentro de organizações com necessidade em melhorar suas rotinas interna de seu software”, os motivou na criação de sua obra, certamente carregarei essa motivação para realização deste

estudo.

1.4 OBJETIVO GERAL DO TRABALHO

Propor uma sistemática para implantação dos procedimentos e para o gerenciamento dos processos administrativos no GRUPO FRANCO BRASIL, buscando a melhoria no seu sistema de gestão de uma empresa hospitalar do estado do Ceará.

1.5 OBJETIVOS ESPECÍFICOS DO TRABALHO

- Verificar a melhor sistemática de implantação;
- Diagnosticar a estrutura da empresa hospitalar nos aspectos: organizacional, de pessoas e das divisões de trabalho - processos;
- Implementar processos em cada setor estruturado para o grupo;
- Averiguar a importância do desenvolvimento do software e suas principais características;
- Responder como organizar o desenvolvimento de rotinas do software.

1.6 RESULTADOS ESPERADOS DA PESQUISA

Um dos resultados esperados com este trabalho é a otimização dos processos realizados em cada setor, garantindo a padronização, acompanhamento e melhoria contínua, conforme as reais necessidades da empresa.

Também se espera fornecer uma nova fonte de pesquisa e estudo para a área, e ao final deixar uma nova idéia de organização ou possibilitar, despertar curiosidades de futuras pesquisas caso ou não o problema de pesquisa seja sanado. Segundo (FRANCO, 2020, p. 12) “ao final do estudo” há uma possibilidade de “apresentar um perfil e uma nova fonte de pesquisa”, contudo, “no final, fornecer dados empíricos que possibilitem aprofundar mais esta temática” de desenvolvimento, dessas rotinas dos (SLD) sistemas legados distribuídos organizacional.

1.7 ESTRUTURA DO ARTIGO PARA PESQUISA

Este artigo está estruturado em cinco capítulos todos dividido em partes para melhor entendimento e organização do tema. O primeiro refere-se à introdução onde

são apresentadas as motivações, o problema de pesquisa, os objetivos e a justificativa. O capítulo 2 agarrar-se à revisão da literatura com os assuntos basilares do artigo. As questões metodológicas são discutidas no capítulo 3. No capítulo 4 para além da apresentação dos dados coletados, faz-se a respectiva discussão e análise. Por fim, no capítulo 5, são apresentadas considerações finais, conclusões, limitações e sugestões para trabalhos futuros.

2 **DESENVOLVIMENTO**

Houve uma duvida e dessa duvida surgiu algo possível a ser planejado e foi um pensamento em unir algumas idéias de grandes escritores e seus diversos temas voltados para a área da engenharia de software que tornou possivelmente o despertar e a curiosidade de pesquisar e escrever este artigo, que provavelmente servirá como ferramenta de auxilio para futuras gestões e modelos de padrões de desenvolvimento de software. Contudo provavelmente essa pesquisa busque preencher lacunas que até o presente momento não foram preenchidas que possa ser a centralização de varias idéias e trabalhos científicos em um só, que este seja um trabalho completo e repleto de opiniões centralizadas e que sirva de fonte de pesquisa futuras.

2.1 REVISÃO DA LITERATURA DA PESQUISA

Para levar a empresa à excelência, o administrador deve ter espírito empreendedor, aceitar desafios, assumir riscos e possuir um senso de inconformismo sistemático, para conduzir a empresa a uma situação melhor (CHIAVENATO, 2004).

Segundo (CHIAVENATO, 2014, p. 8), o profissional que se dedica a administrar as habilidades práticas e concretas de como fazer e executar certas coisas de maneira correta e eficiente para atividades administrativas, em partes, orientadas para o campo do diagnóstico e da decisão em que utiliza suas habilidades conceituais de perceber e definir situações e equacionar estratégicas de ação adequadas e eficazes para aquelas situações, entretanto, possibilita a maior necessidade de se fundamentar em conceitos, idéias, teorias e valores que lhe permitam a orientação no quesito profissional e o balizamento de seu comportamento de todos aqueles que trabalham sob sua direção ou orientação. Segue-se, portanto, a teoria geral da administração é orientadora do comportamento, busca ensinar o que deve ser feito em determinadas circunstâncias ou ambiente.

O cenário organizacional é formado por circunstâncias originadas tanto no ambiente externo, segundo uma visão macro, quanto no ambiente interno, numa perspectiva micro (ARAUJO; GARCIA; MARTINES, 2017).

Segundo (CHIAVENATO, 2014), desde a Revolução Industrial e das

metodologias propostas por Taylor, pode-se observar a preocupação da departamentalização da empresa, no intuito de segmentar e controlar cada atividade da organização. Nos dias atuais, percebe-se a importância maior da padronização do processo para uma eficiência no resultado da empresa.

Segundo (ARAUJO; GARCIA; MARTINES, 2017) sobre a teoria geral, citam Ludwig Von Bertalanffy, biólogo alemão que elaborou a Teoria Geral dos Sistemas (TGS); onde, os sistemas só podem ser compreendidos se trabalhados de forma integrada, argumentando-se como base os sistemas do corpo humano que coexistem e interagem impactando diretamente na saúde (resultado). Trazendo para a realidade organizacional, segundo a TGS, não é possível trabalhar cada departamento, setor, seção de forma isolada. Pelo contrário, é necessário analisar o todo organizacional, porque, muitas vezes, o problema está na relação entre as unidades e não dentro delas (ARAUJO; GARCIA; MARTINES, 2017).

Para (BERTALANFFY, 2010) os processos parciais podem ser sobrepostos para obter o processo total. Entretanto, o problema metodológico da teoria dos sistemas consiste, portanto em preparar-se para resolver problemas.

Segundo (PALADINI, 2010), a atenção ao processo produtivo foi um estágio posterior do desenvolvimento da Gestão da Qualidade em sua totalidade. Por muito tempo, a qualidade era avaliada em produtos e serviços, voltando sua atenção para resultados de atividades ou efeitos de ações bem definidas. Buscava-se, portanto, conferir confiabilidade à análise da qualidade do produto, pois havia o entendimento de que essa era a forma pela qual o cliente avaliava toda a empresa. Todo o esforço, assim, visava à qualidade do produto acabado e essa era uma forma rudimentar de entender os padrões da qualidade adotados pelo cliente.

O processo produtivo visa, de fato, a atender as necessidades do cliente, a razão de ser da empresa. Porém, não atentando para os processos internos que interferem, também, no resultado da empresa, pode, a mesma, apresentar produtos satisfatórios ao cliente e não ter condições de acompanhar ou avaliar os resultados da mesma. Pode também, haver perdas financeiras fora do âmbito produtivo, tornando-a satisfatória para os clientes externos, porém, sem lucratividade que justifique ou permita a sua existência.

Pode ser que, a evolução das ferramentas de desenvolvimento tenha afetado de alguma maneira a produção e qualidade no processo de criação, melhorias de

novas rotinas dos sistemas legados das organizações, vindo a causar um forte crescimento no processo de desenvolvimento dessas rotinas do software. Segundo (FREITAS, 2015, p. 14) “essas ferramentas evoluíram ainda mais permitindo uma maior qualidade nos programas, bem como uma produtividade mais alta por parte dos programadores”. Na “época o conceito de engenharia de software começou a ser discutido tendo como propósito aplicar os conceitos de engenharia ao desenvolvimento de sistemas de software complexos”.

2.2 SOBRE A PADRONIZAÇÃO DE PROCESSOS

Processo administrativo é o nome dado ao conjunto de funções administrativas, que envolvem o planejamento, a organização, a direção e o controle (CHIAVENATO, 2014, p. 95).

Nas empresas modernas, a padronização é considerada a mais fundamental das ferramentas gerenciais. Segundo (CAMPOS, 2013), os gerentes precisam entender que a padronização é um dos caminhos seguros para a produtividade e competitividade em nível internacional, pois é uma das bases onde se assenta o moderno gerenciamento. Na Qualidade Total, ela é a base para a Rotina (Gerenciamento da Rotina do Trabalho Diário), e é meio, onde o objetivo é conseguir melhores resultados.

A padronização deve ser vista como um processo interno e contínuo da empresa, o qual proporcione melhorias em qualidade, custo, cumprimento de prazo e segurança nos processos. Ela também facilita o treinamento de novos integrantes na função a ser exercida por eles, uma vez que se tem o projeto e os procedimentos utilizados para atingir o objetivo.

Assim, pode proporcionar às empresas uma maior confiança nos processos, não pondo uma dependência da organização nas mãos de funcionários. Ao reunir as pessoas e discutir o procedimento, até encontrar aquele que for melhor, é possível treiná-las e assegurar-se de que a execução está de acordo com o que foi acordado. Dessa forma, o acompanhamento das atividades é mais claro e transparente, sendo verificado seu cumprimento através de índices de controle.

A padronização garante que o processo não seja alterado, sendo cumprido por todos os envolvidos no processo. Porém, a importância está voltada não somente para o processo, como para as pessoas, o que permite que os

procedimentos sejam revistos à medida que a necessidade surja, criando-se, assim, um ciclo de melhoria contínua a partir da padronização do processo – princípio do conceito da Qualidade Total. Daí a importância de se entender que, pela padronização decorrem a educação, o treinamento e a delegação, para que esse ciclo esteja sempre de acordo com os objetivos da empresa.

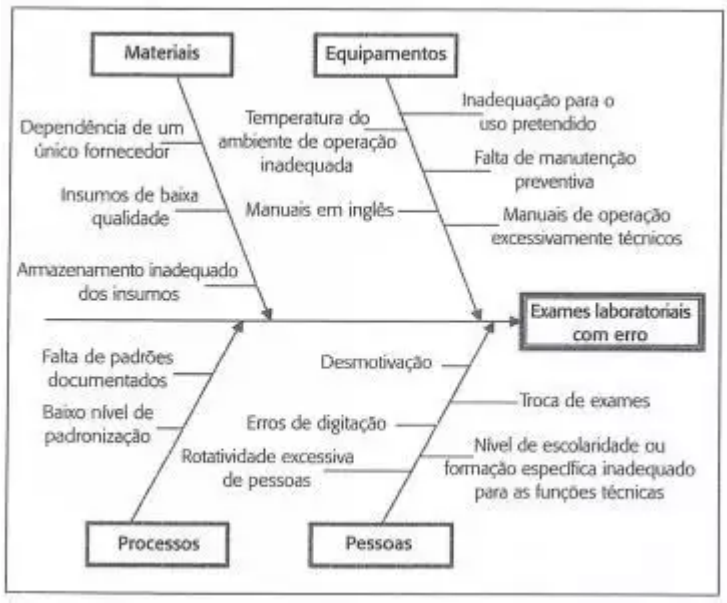
Segundo (MOREIRA, 2013), a qualidade é entendida, normalmente, como um atributo de produtos ou serviços, mas pode referir-se a tudo que é feito pelas pessoas; fala-se na qualidade de um aparelho elétrico, de um carro, do serviço prestado por um hospital, do ensino provido por uma escola, ou do trabalho de um dado funcionário ou departamento.

Continuando com (MOREIRA, 2013), o Total Quality Management (TQM) é uma filosofia gerencial, isto é, uma forma particular de enxergar uma instituição, para que ela serve e como administrá-la. Esta é uma filosofia integrada por gerência e por um conjunto de práticas que enfatizam a melhoria contínua, a busca pelo atendimento das necessidades do cliente, o pensamento em longo prazo, a eliminação de refugo e retrabalho, o envolvimento do trabalhador, o trabalho em equipe, os novos projetos do processo, o benchmarking (busca e adoção das melhores práticas conhecidas de trabalho), a análise e solução de problemas pelos empregados, a medida de resultados e o relacionamento próximo com os fornecedores.

A palavra processo pode ser verificada e analisada em todos os aspectos e setores da empresa. O objetivo principal do projeto de processos é assegurar que o seu desempenho seja adequado ao que quer que se esteja tentando alcançar (SLACK; CHAMBERS; JOHNSTON, 2009).

No levantamento das informações de todo o processo, para definição do padrão a ser estabelecido, pode-se utilizar a metodologia utilizada na ferramenta chamada Diagrama de Causa e Efeito ou Diagrama de Ishikawa (seu criador), que analisa os itens necessários para se ter uma visão ampla do processo. É uma ferramenta de representação das possíveis causas que levam a um determinado efeito a ser analisado, conforme Figura 1 representada a seguir:

Figura 1 — Diagrama de causa e efeito



Fonte: Adaptado de Marshall et al. (2015)

2.3 PONTOS RELACIONADOS DA PESQUISA

Possivelmente a idéia de organizar as rotinas do software de uma organização seria possível na medida em que a pesquisa demonstre pontos que possamos acreditar que sejam relevantes para serem aplicados na pratica. Poderíamos imaginar ou acreditar que a centralização do código da aplicação é uma maneira de organizá-lo? Responderíamos que sim ou não, mas caso a resposta chegue a ser sim, então seria uma forte possibilidade de acreditarmos em um novo propósito para organizar e centralizar as rotinas, onde poderá ser representado pelo o modelo da simples seqüência (PDSS) propósito proposto para um controle organizado e centralizado de rotinas. Para (SOMMERVILLE, 2016, p. 224) “os sistemas de software não são sistemas isolados, mas componentes essenciais de sistemas mais abrangentes com algum propósito humano, social ou organizacional”.

2.3.1 **Sobre preferências de implementação do software**

Será que escolhe entre uma abordagem ou outra tornará diferente a idéia central do propósito? Mudará o jeito de como a informação e implementação da rotina é tratada nos padrões já existentes? Será que, o que está sendo apresentado nesse trabalho pode vim a preencher lacunas que falta e complementar essa fonte de estudo? Esse propósito organizado, centralizado de rotinas que visa um controle através do modelo da simples seqüência (PDSS) pode ser dito como uma nova

abordagem? Segundo (SOMMERVILLE, 2011, p. 403) “o sistema central é um conjunto de recursos do sistema que implementa seu propósito essencial. Portanto, se o propósito de determinado sistema é manter as informações, o sistema central fornece meios para criar, editar, gerenciar e acessar um banco de dados de registros”.

2.3.1.1 Das características fundamentais do software

Seria sensato talvez disser que devemos aplicar na pratica a idéia de melhoria da engenharia de desenvolvimento do software, levando sempre em consideração o que ser lê ver e aprende no dia a dia. Segundo (SOMMERVILLE, 2016, p. 16) “a engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software”, esse talvez seja a maior razão desse trabalho, levar essa idéia até o final do mesmo, essa engenharia porem apoia o desenvolvimento profissional “mais do que a programação individual”. Acredita-se em características fundamentais que esse objetivo possa estar dentro dessas características, ou seja, um forte ponto fundamental. Continuando a referencia de engenharia de software, “ela inclui técnicas que apoiam especificação, projeto e evolução de programas, que normalmente não são relevantes para o desenvolvimento de software pessoal”.

2.3.1.1.1 *Da representação do propósito*

Possivelmente poderíamos representar o nosso modelo da simples seqüência o PDSS (propósito para controle organizado, centralizado de rotinas) como um complemento dos padrões de engenharia já adotado hoje em dia, o intuito do propósito desse trabalho não é centrar o desenvolvimento das rotinas de um software a essa idéia e sim fazer uma possível complementação as idéias já adotadas e apresentá-la hoje no século XXI, que possa tornar possível com esse recurso uma melhor qualidade no processo de desenvolvimento para o profissional e empresa que venha a praticar essa representação.

2.4 DA SIMPLES SEQUÊNCIA DE CONTROLE DE ROTINA

Para falamos do modelo de organização e centralizador de rotina da simples seqüência, primeiro precisamos falar pelo menos de um modelo já existente como, por exemplo, o modelo cascata, que o foco seria antes de iniciar o projeto, nesse caso projeta-se todas as etapas do processo, para (SOMMERVILLE, 2016, p. 36), “não existe um modelo de processo universal certo para todos os tipos de

desenvolvimento de software". Portanto, "o processo certo depende do cliente e da regulamentação dos requisitos, o ambiente onde o software será usado, e o tipo de software produtos em desenvolvimento". Portanto exemplifica que, "o software crítico de segurança geralmente é desenvolvido usando um processo em cascata, pois muitas análises e documentação são necessárias antes a implementação começa". Então pressuponha que o novo propósito e essa nova idéia do modelo terão lógicos e com as bases apresentadas nesse trabalho virá a preencher as lacunas que hoje temos na vida real dentro das organizações, onde foi citado. Mas será mesmo que "não existe um modelo de processo universal certo para todos os tipos de desenvolvimento de software"? Continuando nas palavras de sommerville "produtos de software agora são sempre desenvolvidos usando um modelo de processo mental. Os sistemas de negócios estão cada vez mais sendo desenvolvidos por descobrir os sistemas existentes e integrá-los para criar um novo sistema com a funcionalidade que é necessária". Já o foco do modelo está no meio e fim e não no antes, por isso é necessário adotar todos os padrões já existentes de engenharia de software para implementar o modelo. Nota-se (SOMMERVILLE, 2016, p. 36) que, "a maioria dos processos de software práticos são baseados em um modelo geral", continuando, "mas freqüentemente incorporam recursos de outros modelos". Fazendo sentido quando descrevo que é essencial adotá-lo junto com os modelos já existentes. Portanto, "isso é particularmente verdadeiro para grandes sistemas de engenharia". Certamente o modelo pode vim há ter uma base a ser sustentada pela a existência de outros modelos padrões já adotada nas organizações.

2.4.1 **Visual do possível modelo do propósito**

Segundo (SOMMERVILLE, 2016, p. 658) "se uma organização estiver em um estado de turbulência com constantes reorganizações e insegurança no trabalho, é difícil para os membros da equipe foco no desenvolvimento de software". Então há uma possibilidade de tira a equipe de desenvolvimento do seu foco principal, mas esse modelo poderá auxiliar na organização desta ou de possíveis turbulências.

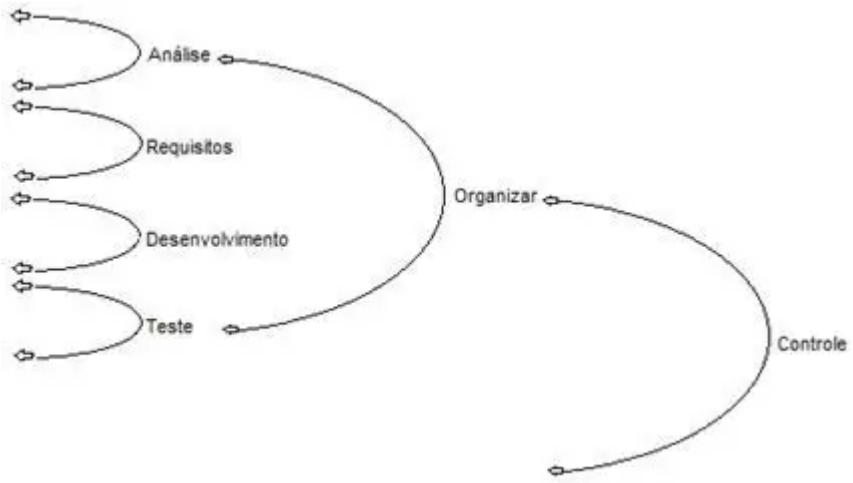
Lembrando que esse modelo ele pode estar ligado ou desligado, acoplado ou desacoplado dos processos, ou seja, podem utilizá-lo para adequação junto aos processos já existentes ou não. A representação partiu-se de uma vivencia e experiência na área de desenvolvimento de rotinas de software e legado, a idéia surgiu em busca de descobrir soluções e a melhor maneira de atuar nos atrasos de desenvolvimento e requisitos mal repassados, e o não conhecimento das funcionalidades já existentes no sistema legado ou software da organização por

parte dos usuários dos diversos setores da organização, para (SOMMERVILLE, 2016, p. 24) “sempre tente descobrir soluções para problemas, mesmo quando não houver aplicativos teorias e métodos de cabos”. Essas palavras são encorajadoras, e há uma forte ligação com os motivos aqui citados, continuando, “os engenheiros também reconhecem que devem trabalhar dentro das restrições organizacionais e financeiras, e eles devem procurar soluções dentro dessas restrições”.

Segundo (SOMMERVILLE, 2016, p. 24), “em geral, os engenheiros de software adotam uma abordagem sistemática e organizada para funciona, pois geralmente é a maneira mais eficaz de produzir software de alta qualidade”. Porém, “a engenharia trata de selecionar o método mais adequado para um conjunto de circunstâncias, então uma abordagem mais criativa e menos formal para o desenvolvimento pode ser o certo para alguns tipos de software”.

Para (SOMMERVILLE, 2016, p. 47), “um modelo de processo de software (às vezes chamado de ciclo de vida de desenvolvimento de software ou modelo SSDLC) é uma representação simplificada de processo de software. Cada modelo de processo representa um processo de uma perspectiva particular e, portanto, fornece apenas informações parciais sobre esse processo”. Com base nessas palavras farei logo abaixo a representação de uma perspectiva particular de um modelo de processo de software.

Figura 2 — Representação genérica do propósito da simples sequência (PDSS)



Fonte: O autor (2021)

Observaremos a figura acima, pretende-se fazer desse modelo semelhante a

decida de uma montanha em ritmo de pulo com os pés juntos, de sentido único para baixo mais com possibilidades de iniciar novamente a fazer anterior, onde é feita uma parada para organizar nossas rotinas. Entretanto, pretende-se iniciar sempre com a organização, onde poderemos descer sempre testando as funcionalidades das rotinas e verificando os requisitos e nunca iniciamos o desenvolvimento sem ter, testado as rotinas antigas e requisito, assim do mesmo modo para a análise. Segundo (SOMMERVILLE, 2016, p. 50), “o processo de software, na prática, nunca é um modelo linear simples, mas envolve alimentação de voltar de uma fase para outra”. Esse modelo é para ser utilizado com os modelo já existentes, devemos acopla-lo a nova fase ou etapa corrente do processo. Deveremos liga-lo com uma seta a etapa do processo, quando se liga, recebemos as informações da etapa do processo e organizamos e após essa organização seguiremos com o fluxo proposto do modelo, quando se tem uma estabilidade da etapa do processo poderemos descer os degraus até o nível de organização, organizar as informações das rotinas e desacopla ou desligar nosso modelo, devolvendo todas as informações organizadas do desenvolvimento. Para (SOMMERVILLE, 2016, p. 50), “na medida em que novas informações surgem em uma etapa do processo, os documentos produzidos em nossos estágios devem ser modificados para refletir as mudanças necessárias no sistema”. Porém, ”por exemplo, se for descoberto que um requisito é muito caro para implementar, os requisitos documento deve ser alterado para remover esse requisito”. Mas, “no entanto, isso requer aprovação do cliente e atrasa o processo geral de desenvolvimento”.

Porque esse modelo de organizar e centralizar são necessários? Acredita-se que muitas empresas não se adequam aos modelos atuais de processo de software, as etapas apresentam falhas e erros são descobertos na fase final de todo o processo, ou até mesmo por seus profissionais não conhecerem o processo de engenharia de software ou por desconhecerem as inúmeras rotinas atuais do sistema. Segundo (SOMMERVILLE, 2016, p. 54), “como resultado, tanto clientes quanto desenvolvedores podem congelar prematuramente o software especificação de modo que nenhuma alteração adicional seja feita a ela”. Isso é um fato, “infelizmente, isso significa que os problemas são deixados para resolução posterior, ignorados ou programados”. Entretanto, “prematuro o congelamento de requisitos pode significar que o sistema não fará o que o usuário deseja”. Contudo, “isto também pode levar a sistemas mal estruturados, pois os problemas de design são contornados por truques de implementação”.

Acredita-se que a falhas de erro de desenvolvimento estar ligada diretamente

as etapas de requisito e testes por isso a ideia e foco desse modelo estar centrada nessas etapas. Segundo (SOMMERVILLE, 2016, p. 50), ao se referir da “integração e teste do sistema”, ressalta que “as unidades de programa ou programas individuais são integrado e testado como um sistema completo para garantir que o software requisitos foram cumpridos”. Entretanto, “após o teste, o sistema de software é entregue para o consumidor”. Nota-se ainda que “operação e manutenção normalmente, esta é a fase do ciclo de vida mais longa”. Contudo “o sistema é instalado e colocado em uso prático. A manutenção envolve correção erros que não foram descobertos nas fases anteriores do ciclo de vida”, assim “melhorando a implementação de unidades do sistema e aprimoramento dos serviços do sistema como novos requisitos são descobertos”.

Foi pesando em todas essas abordagem que acredita-se que esse modelo organizador e centralizador de rotinas da simples sequência possa evitar erros futuros ou até prevenir surgimentos desses na fase final do processo. Segundo (SOMMERVILLE, 2016, p. 51), “durante a fase final do ciclo de vida (operação e manutenção), o software é colocado em uso”. Porém, “erros e omissões nos requisitos de software originais são descobertos”. Contudo, “surgem erros de programa e design, e a necessidade de novas funcionalidades é identificada”. Entretanto, “o sistema deve, portanto, evoluir para permanecer útil. Fazendo essas mudanças (software manutenção) pode envolver a repetição de etapas anteriores do processo”. Contudo, “na realidade, o software deve ser flexível e acomodar as mudanças como estão sendo desenvolvido”. Entretanto, “a necessidade de comprometimento antecipado e retrabalho do sistema quando as mudanças são feito significa que o modelo em cascata é apropriado apenas para alguns tipos de sistema”.

2.4.1.1 Dos modelos

Se necessário antes de ligar a idéia apresentada, seguir as existentes. Segundo (SOMMERVILLE, 2016, p. 48), “não existe um modelo de processo universal certo para todos os tipos de desenvolvimento de software”. Porém, “o processo certo depende do cliente e da regulamentação requisitos, o ambiente onde o software será usado, e o tipo de software produtos em desenvolvimento”. Mas, ”por exemplo, o software crítico de segurança geralmente é desenvolvido usando um processo em cascata, pois muitas análises e documentação são necessárias antes a implementação começa”.

2.4.1.2 Modelo cascata

Nota-se (SOMMERVILLE, 2016, p. 47), que “o modelo em cascata Este leva as atividades de processo fundamentais de especificação, desenvolvimento”, entretanto essa “validação e evolução e os representa separadamente fases do processo, como especificação de requisitos, design de software, implementação e teste”.

2.4.1.3 Sobre o desenvolvimento incremental

Segundo (SOMMERVILLE, 2016, p. 48), “desenvolvimento incremental esta abordagem intercala as atividades de especificação, desenvolvimento e validação”. Contudo, “o sistema é desenvolvido como uma série de versões (incrementos), com cada versão adicionando funcionalidade à versão anterior”.

2.4.1.4 Da integração e configuração de forma pratica

Para (SOMMERVILLE, 2016, p. 48), “Integração e configuração Esta abordagem depende da disponibilidade de reutilização componente ou sistemas capazes”. Entretanto, “o processo de desenvolvimento do sistema se concentra em configurar esses componentes para uso em uma nova configuração e integrá-lo sem um sistema”.

Segundo (SOMMERVILLE, 2016, p. 55-56), a “engenharia de software orientada para a reutilização, baseada em configuração e integração, tem a vantagem óbvia de reduzir a quantidade de software a ser desenvolvido e assim reduzindo custos e riscos”. Contudo, “geralmente também leva a uma entrega mais rápida do software. No entanto, o comprometimento de requisitos é inevitável, e isso pode levar a um sistema que não atende às reais necessidades dos usuários”. Entretanto, “além disso, algum controle sobre o sistema evolução de tempo é perdida, pois novas versões dos componentes reutilizáveis não estão sob o controle da organização que os utiliza”. Mas, “contudo ressalta que a reutilização de software é muito importante”.

2.4.2 **Descrevendo os fundamentos de análise**

Para (SOMMERVILLE, 2016, p. 31), “engenheiros de software devem se comprometer a fazer a análise, especificação, projeto, desenvolvimento, teste gerenciamento e manutenção de software uma profissão benéfica e respeitada”.

Segundo (SOMMERVILLE, 2016, p. 48), “muitas análises e documentação são necessárias antes a implementação começa”.

Nota-se (SOMMERVILLE, 2016, p. 49), a “análise e definição de requisitos os serviços do sistema, restrições e as metas são estabelecidas por meio de consulta aos usuários do sistema”. Porém, “eles são então definidos em detalhes e servir como uma especificação do sistema”.

Para (SOMMERVILLE, 2016, p. 52), “o custo de implementação de mudanças de requisitos é reduzido”. Entretanto, “a quantidade de análise e documentação que precisa ser refeita é significativamente menor do que exigido com o modelo em cascata”.

2.4.3 **Descrevendo os fundamentos de desenvolvimento**

Segundo (SOMMERVILLE, 2016, p. 24), “todos os aspectos da produção de software a engenharia de software não está apenas preocupada com os processos técnicos de desenvolvimento de software”. Contudo, “também inclui atividades como gerenciamento de projetos de software e desenvolvimento de ferramentas, métodos, e teorias para apoiar o desenvolvimento de software”.

Para (SOMMERVILLE, 2016, p. 19), “desenvolvimento de diferentes tipos de software sistema pode exigir diferentes técnicas de engenharia de software”.

Segundo (SOMMERVILLE, 2016, p. 20), a “engenharia de software é criticada como inadequada para desenvolvimento de software moderno”. No entanto, em sua opinião, muitas dessas chamadas falhas de software são conseqüência de dois fatores:

> 1. Aumento da complexidade do sistema à medida que novas técnicas de engenharia de software nos ajudam para construir sistemas maiores e mais complexos, as demandas mudam. Os sistemas têm que ser construído e entregue mais rapidamente; sistemas maiores e ainda mais complexos são requeridos; e os sistemas precisam ter novos recursos que antes eram pensados para ser impossível. Novas técnicas de engenharia de software devem ser desenvolvido para atender aos novos desafios de entrega de software mais complexo.
>
> 2. Falha ao usar métodos de engenharia de software é bastante fácil escrever para computador programas sem usar métodos e técnicas de engenharia de software. Muitos empresas migraram para o desenvolvimento de software à medida que seus produtos e serviços os vícios evoluíram.

> Eles não usam métodos de engenharia de software em todos os seus dia de trabalho. Conseqüentemente, seu software é freqüentemente mais caro e menos confiável do que deveria ser. Precisamos de melhor educação e treinamento em engenharia de software para resolver este problema (SOMMERVILLE, 2016, p. 20).

Nota-se (SOMMERVILLE, 2016, p. 20), que, “os engenheiros de software podem se orgulhar de suas realizações” e feitos. Contudo é bom lembrar, “ainda temos problemas para desenvolver software complexo”, entretanto “sem engenharia de software nós não teríamos explorado o espaço e não teríamos a Internet ou a televisão com comunicações modernas”.

2.4.4 **Descrevendo os fundamentos de requisitos**

Para (SOMMERVILLE, 2016, p. 104), “os requisitos para um sistema são as descrições dos serviços que um sistema deve fornecer e as restrições ao seu funcionamento”. Entretanto, “esses requisitos refletem as necessidades de clientes para um sistema que serve a um determinado propósito”, por exemplo, “como controlar um dispositivo, colocar um pedido ou localização de informações”. Por tanto, “o processo de descobrir, analisar, documentar e a verificação desses serviços e restrições é chamada de engenharia de requisitos (RE)”. Contudo, “o termo requisito não é usado de forma consistente na indústria de software”. Entretanto, “em alguns casos, um requisito é simplesmente uma declaração abstrata de alto nível de um serviço que um sistema deve fornecer ou uma restrição em um sistema. No outro extremo, é um definição formal e detalhada de uma função do sistema”.

Segundo (SOMMERVILLE, 2016, p. 104), “alguns dos problemas que surgem durante o processo de engenharia de requisitos são resultado de não conseguir fazer uma separação clara entre esses diferentes níveis de descrição”. Porém, seria possível, usar o “termo requisitos do usuário para significar o requisitos abstratos de alto nível e requisitos de sistema para significar o detalhado descrição do que o sistema deve fazer”. Contudo, “requisitos do usuário e requisitos do sistema podem ser definidos da seguinte forma”:

> 1. Os requisitos do usuário são declarações, em uma linguagem natural mais diagramas, de quais serviços vícios que o sistema deve fornecer aos usuários do sistema e as restrições sob que deve operar. Os requisitos do usuário podem variar de declarações gerais dos recursos do sistema necessários para descrições detalhadas e precisas da funcionalidade do sistema.

> 2. Os requisitos do sistema são descrições mais detalhadas do sistema de software funções, serviços e restrições operacionais. O documento de requisitos do sistema mento (às vezes chamado de especificação funcional)

> deve definir exatamente o que é ser implementado. Pode ser parte do contrato entre o comprador do sistema e os desenvolvedores de software (SOMMERVILLE, 2016, p. 104).

Para (SOMMERVILLE, 2016, p. 105), “os requisitos do sistema fornecem informações mais específicas sobre os serviços e funções do sistema a ser implementado”. Porém, “precisa-se escrever requisitos em diferentes níveis de detalhe porque diferentes tipos de leitores os usam de maneiras diferentes”. Entretanto, “os leitores dos requisitos do usuário não são usualmente preocupados como o sistema será implementado e podem ser gerentes que não estão interessados nas instalações detalhadas do sistema”. Contudo, “os leitores do sistema de requisitos precisam saber com mais precisão” e com riqueza de detalhes “o que o sistema fará porque eles são preocupados em como isso irá apoiar os processos de negócios ou porque eles estão envolvidos na implementação do sistema”.

Segundo (SOMMERVILLE, 2016, p. 106), “a engenharia de requisitos geralmente apresentada como o primeiro estágio do processo de engenharia do software”. Possivelmente o processo essencial para um bom desenvolvimento, ”no entanto, alguma compreensão dos requisitos do sistema pode devem ser desenvolvidos antes que uma decisão seja tomada para ir em frente com a aquisição ou desenvolvimento de um sistema”.

2.4.5 **Descrevendo os fundamentos de testes**

Nota-se (SOMMERVILLE, 2016, p. 60), “a programação é uma atividade individual e não existe um processo geral que seja geralmente seguido”. Entretanto, “alguns programadores começam com componentes que entendem, desenvolva-os e, em seguida, passam para os componentes menos compreendidos”. Contudo, “outros tomam uma abordagem o posta, deixando os componentes familiares por último porque eles sabem como desenvolvê-los”. Porém, “alguns desenvolvedores gostam de definir os dados no início do processo e depois usa isso para conduzir o desenvolvimento do programa, outros deixam dados não especificados como tanto quanto possível”. Por tanto, “normalmente, os programadores realizam alguns testes do código que desenvolveram. Isso geralmente revela defeitos do programa (bugs) que devem ser removidos do programa”. Entretanto, “a localização e correção de defeitos do programa é chamada de depuração . Teste de defeito e depuração são processos diferentes”. Porém, “o teste estabelece a existência de defeitos. Depurando-os preocupa-se em localizar e corrigir esses defeitos”.

Segundo (SOMMERVILLE, 2016, p. 60), “quando depurando, deve-se gerar hipóteses sobre observar comportamento do programa e, em seguida, testar essas hipóteses na esperança de encontrara falha que causou a anomalia de saída”. Entretanto, “o teste das hipóteses pode envolver rastreamento do código do programa manualmente. Pode exigir novos casos de teste para localizar o problema”. Por tanto, “ferramentas de depuração interativas, que mostram os valores intermediários do programa variáveis e um traço das instruções executadas, geralmente são usados para apoiar o processo de depuração”.

Segundo (SOMMERVILLE, 2016, p. 60), “a validação de software ou, mais geralmente, a verificação e validação (V & V) pretende mostrar que um sistema está em conformidade com sua especificação e atende a expectativas do cliente do sistema”. Entretanto, “teste de programa, onde o sistema é executado usando dados de teste simulados, é a principal técnica de validação”. Por tanto, “a validação também pode envolvem processos de verificação, como inspeções e revisões, em cada etapa do processo de software desde a definição dos requisitos do usuário até o desenvolvimento do programa”. Porém, “no entanto, a maior parte do tempo e do esforço da V&V é gasta em testes de programas”. Contudo, “exceto para pequenos programas, os sistemas não devem ser testados como um único, monolítico unidade”. Entretanto, “para o software personalizado, o teste do cliente envolve o teste do sistema com dados reais do cliente”. Por tanto, “para produtos que são vendidos como aplicativos, o teste do cliente às vezes é chamado de teste beta, onde usuários selecionados experimentam e comentam sobre o software”. Por tanto seguindo com o assunto de processo de testes do sistema, “as etapas do processo de teste são”:

> 1. Teste de componentes os componentes que compõem o sistema são testados pelas pessoas desenvolver o sistema. Cada componente é testado independentemente, sem outros componentes do sistema. Os componentes podem ser entidades simples, como funções ou classes de objetos ou podem ser agrupamentos coerentes dessas entidades.
>
> 2. Teste do sistema os componentes do sistema são integrados para criar um sistema completo. Este processo se preocupa em encontrar erros resultantes de imprevistas interações entre componentes e problemas de interface de componentes. Isso é também preocupada em mostrar que o sistema atende às suas funções funcionais e não funcionais requisitos e testando as propriedades do sistema emergente. Para grandes sistemas, este pode ser um processo de vários estágios onde os componentes são integrados para formar subsistemas que são testados individualmente antes que esses subsistemas sejam integrados a formar o sistema final.
>
> 3. Teste do cliente este é a fase final do processo de teste antes do sistema é aceito para uso operacional. O sistema é testado pelo cliente do sistema (ou cliente potencial) em vez de dados de teste simulados. Para

> customização do software, o teste do cliente pode revelar erros e omissões no sistema definição de requisitos, porque os dados reais exercem o sistema em diferentes maneiras a partir dos dados de teste. O teste do cliente também pode revelar problemas de requisitos onde as instalações do sistema não atendem realmente às necessidades dos usuários ou o sistema o desempenho é inaceitável. Para produtos, o teste do cliente mostra quão bem o produto de software atende às necessidades do cliente (SOMMERVILLE, 2016, p. 61).

2.4.6 **Descrevendo os fundamentos da mudança**

Para (SOMMERVILLE, 2016, p. 24), “o software deve ser escrito de tal forma que possa evoluir para atender às necessidades de mudança dos clientes”. Porém, “este é um atributo crítico porque a mudança de software é um requisito inevitável de um ambiente de negócios em mudança”.

Segundo (SOMMERVILLE, 2016, p. 63), “a mudança é inevitável em todos os grandes projetos de software”. Por tanto, “os requisitos do sistema mudam conforme as empresas respondem a pressões externas, concorrência e mudanças prioridades de gestão”. Porém, “na medida em que novas tecnologias se tornam disponíveis, novas abordagens para design e implementação tornam-se possíveis”. Com essa disponibilidade, “portanto, qualquer software pro modelo de processamento é usado, é essencial que ele possa acomodar alterações no software sendo desenvolvido”. Nota-se (SOMMERVILLE, 2016, p. 63), “mudança aumenta os custos de desenvolvimento de software porque geralmente significa aquele trabalho que foi concluído deve ser refeito”. Entretanto, “isso é chamado de retrabalho”. Entretanto, “para exemplo, se os relacionamentos entre os requisitos em um sistema foram analisados e novos requisitos são então identificados, alguns ou todos os requisitos a análise deve ser repetida”. Contudo, “pode ser necessário redesenhar o sistema para entregar os novos requisitos, alterar quaisquer programas que tenham sido desenvolvidos, e teste novamente o sistema”. Existem, “duas abordagens relacionadas que podem ser usadas para reduzir os custos de retrabalho”:

> 1. Mudança de antecipação, onde o processo de software inclui atividades que podem antecipar ou prever possíveis mudanças antes que um retrabalho significativo seja necessário. Para exemplo, um sistema de protótipo pode ser desenvolvido para mostrar algumas características-chave de o sistema aos clientes. Eles podem experimentar o protótipo e refinar seus requisitos antes de se comprometer com altos custos de produção de software.
>
> 2. Mudança de tolerância, onde o processo e o software são projetados para que as mudanças pode ser facilmente feito para o sistema. Isso normalmente envolve alguma forma de aumento desenvolvimento mental. As mudanças propostas podem ser implementadas em incrementos que ainda não foram desenvolvidos. Se isso for impossível, então apenas um

> único incremento (uma pequena parte do sistema) pode ter que ser alterado para incorporar a mudança (SOMMERVILLE, 2016, p. 63).

Segundo (SOMMERVILLE, 2016, p. 64), podem existir “duas maneiras de lidar com a mudança e mudar o sistema requisitos”. A primeira seria a “prototipagem do sistema, onde uma versão do sistema ou parte do sistema é desenvolvido rapidamente para verificar os requisitos do cliente e a viabilidade de decisões de design”. Entretanto, “este é um método de antecipação de mudanças, pois permite aos usuários experimente o sistema antes da entrega e, assim, refine seus requisitos”. Contudo, “o número de propostas de mudança de requisitos feitas após a entrega é, portanto, passiveis de ser reduzido”. A segunda seria a “entrega incremental, onde os incrementos do sistema são entregues ao cliente para comentários e experimentação”. Por tanto, “isso apoia a prevenção de mudanças e tolerância de mudança”. Entretanto, “isso evita o compromisso prematuro com os requisitos para o todo o sistema e permite que as mudanças sejam incorporadas em incrementos posteriores em custo relativamente baixo”.

2.4.7 **Descrevendo os fundamentos de tempo**

Segundo (SOMMERVILLE, 2016, p. 22), “a engenharia de sistema está preocupada com todos os aspectos do computador desenvolvimento de sistemas baseados, incluindo hardware, software e engenharia de processos”. Entretanto, “a engenharia de software faz parte disso mais processo geral”. Porém, “lidando com o aumento da diversidade, demandas por entrega reduzida tempos e desenvolvimento de software confiável”. Contudo, “aproximadamente 60% dos custos de software são custos de desenvolvimento, 40% são custos de teste”. Certamente, “para software personalizado, os custos de evolução muitas vezes excedem custos do desenvolvimento”. Continuando, “embora todos os projetos de software tenham que ser gerenciados profissionalmente e desenvolvidos, diferentes técnicas são apropriadas para diferentes tipos do sistema”.

2.4.8 **Sobre ciclo de vida do software**

Para (SOMMERVILLE, 2016, p. 49), “o processo de desenvolvimento de software contempla uma série de etapas, conforme mostrado”. Entretanto, “por causa da cascata de uma fase para outra, este modelo é conhecido como o modelo em cascata ou ciclo de vida do software”. Contudo, “o modelo em cascata é um exemplo de processo orientado pelo plano”. Porém, “em princípio, pelo menos, se planeja e programa todo o processo atividades antes de iniciar o desenvolvimento

de software”.

2.4.9 **Da refatoração do software**

Segundo (SOMMERVILLE, 2016, p. 348), “refatoração, onde o software está continuamente melhorado, não é visto como uma sobrecarga, mas como uma parte necessária do processo de desenvolvimento”.

Para (SOMMERVILLE, 2016, p. 280), “refatoração é o processo de fazer melhorias em um programa para desaceleração através da mudança”. Porém, “significa modificar um programa para melhorar sua estrutura, reduzir sua complexidade ou torná-lo mais fácil de entender”. Contudo, “refatorar às vezes é considerado limitado ao desenvolvimento orientado a objetos, mas os princípios podem ser fato ser aplicado a qualquer abordagem de desenvolvimento”. Continuando, “ao refatorar um programa, não deve adicionar funcionalidade, mas sim concentrar-se na melhoria do programa”. Porém, “pode, portanto, pensar na refatoração como "manutenção preventiva" que reduz os problemas da mudança futura”.

Nota-se (SOMMERVILLE, 2016), “refatoração é uma parte inerente dos métodos ágeis porque esses métodos são baseados em mudança”. Contudo, “a qualidade do programa pode degradar rapidamente, portanto, desenvolvedores ágeis com freqüência refatorar seus programas para evitar essa degradação”. Porém, “a ênfase no teste de regressão em métodos ágeis, diminui o risco de introdução de novos erros por meio da refatoração”. Entretanto, “qualquer erros que são introduzidos devem ser detectáveis, como testes anteriores bem sucedidos devem então falhe”. Continuando, “no entanto, a refatoração não depende de outras” "atividades ágeis", entretanto há uma possível conseqüência ao aplicar os métodos no desenvolvimento.

Para Fowler (2000, p.9), a “refatoração é o processo de alterar um sistema de software de forma que não altere a o comportamento externo do código ainda melhora sua estrutura interna”. Entretanto, “é uma maneira disciplinada de limpar código que minimiza as chances de introdução de bugs. Em essência, quando você refatorar você é melhorando o design do código depois que ele foi gravado”. Nota se Pressman (2011) que a refatoração é “uma técnica de construção que também é um método para otimização de projetos”. Segundo Sommerville (2011) ao mencionar a refatoração nota se que, “a noção de refatoração, ou seja, melhoria da estrutura e

organização de um programa, também é um importante mecanismo que suporta mudanças".

Segundo (GAMMA et al., 2000, p. 325), Ressalta que "um dos problemas no desenvolvimento de software reutilizável é que, freqüentemente, o software tem que ser reorganizado ou refatorado. Os padrões de projeto ajudam a determinar como reorganizar um projeto e podem reduzir o volume de refatoração que você terá que fazer mais tarde".

> Uma vez que o software atingiu a adolescência e é colocado em serviço, sua evolução é governada por duas necessidades conflitantes: (1) o software deve satisfazer mais requisitos, e (2) o software deve ser mais reutilizável. Comumente, novos requisitos acrescentam novas classes e operações e, talvez, novas hierarquias completas de classes. O software passa por uma fase de expansão para atender novos requisitos. Contudo, isto não pode continuar por muito tempo. Eventualmente, o software irá tornar-se muito inflexível e "endurecido" para permitir mais mudanças. As hierarquias de classes não mais corresponderão a qualquer domínio de problema. Pelo contrário, refletirão muitos domínios de problema, e as classes definirão muitas operações e variáveis de instância não-relacionadas. Para continuar a evoluir, o software deve ser reorganizado por um processo conhecido como refatoração. Esta é a fase na qual freqüentemente se detectam possíveis frameworks. A refatoração envolve separar as classes em componentes especiais e componentes de finalidade genérica, movendo as operações para cima ou para baixo ao longo da hierarquia de classes e racionalizando as interfaces das mesmas. Esta fase de consolidação produz muitos tipos novos de objetos, freqüentemente pela decomposição de objetos existentes e pela utilização da composição de objetos, em vez da herança. Por isso a reutilização de caixa preta substitui a reutilização de caixa branca. A necessidade contínua de satisfazer mais requisitos, juntamente com a necessidade de maior reutilização, faz o software orientado a objetos passar por repetidas fases de expansão e consolidação – de expansão à medida que novos requisitos são atendidos, e de consolidação à medida que o software se torna mais genérico (GAMMA et al., 2000, p. 326).

Para (FOWLER, 2019, p. 47), a "palavra refatoração possui duas definições. Refatoração (substantivo)," consiste em "uma alteração feita na estrutura interna do software para torná-lo mais fácil de entender e mais barato de modificar sem alterar seu comportamento observável." E "refatorar (verbo)," consiste em "para reestruturar o software aplicando uma série de refatorações sem mudar seu comportamento observável". Refatorar é, "no entanto," uma, "ferramenta valiosa, um alicate de prata que ajuda a manter um bom controle do seu código. A refatoração é uma ferramenta que pode e deve ser usada para diversas finalidades".

Segundo (GAMMA et al., 2000) , "um dos problemas no desenvolvimento de software reutilizável é que, freqüentemente, o software tem que ser reorganizado ou refatorado". Nota se, "os padrões de projeto ajudam a determinar como reorganizar

um projeto e podem reduzir o volume de refatoração que você terá que fazer mais tarde”.

2.5 DAS LACUNAS E POSSÍVEIS SOLUÇÕES

Acredita-se na existência de uma possível lacuna no modelo cascata já que uma etapa só se inicia após a outra ser concluída. Segundo (SOMMERVILLE, 2016, p. 50), “em princípio, o resultado de cada fase no modelo em cascata é um ou mais documentos aprovados (“assinados”)”. Contudo, “a fase seguinte não deve começar até a fase anterior terminou”. Possivelmente não existe um modelo global de desenvolvimento de software o que possa existe é um modelo aplicável de processo que foi ou não elaborado de acordo com a área de atuação de seus autores, onde há possibilidades de apresentar falhas no processo de desenvolvimento, que possivelmente pode variar de uma organização para a outra.

Segundo (SOMMERVILLE, 2016, p. 168), “o modelo em cascata embora seja inadequado para a maioria dos tipos de desenvolvimento de software”, muitos “processos de engenharia de sistemas de nível superior são orientados a planos de processos que ainda seguem esse modelo”.

2.6 QUALIDADE NO DESENVOLVIMENTO DE ROTINAS DE SOFTWARE

Para (PÁDUA, 2012) geralmente a qualidade de um produto decorre diretamente da qualidade do processo utilizado na produção dele. Entretanto, note-se que importa aqui a qualidade do processo efetivamente utilizado, não do ‘processo oficial’ que pode eventualmente estar descrito nos manuais da organização. Porém, muitas vezes os processos oficiais não são seguidos na prática, por deficiência de ferramentas, contudo, por falta de qualificação das pessoas, ou porque pressões de prazo levam os gerentes dos projetos a eliminar etapas relacionadas com controle da qualidade.

Para (SOMMERVILLE, 2016) ás vezes, garantia de qualidade significa simplesmente a definição de procedimentos, podendo ter, processos e padrões que visam reforçar que a qualidade de software seja atingida. Entretanto, em outros casos, a garantia de qualidade também inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicadas após o produto terem sido entregue por uma equipe de desenvolvimento.

Segundo (PRESSMAN, 2011) apesar de gerentes e profissionais envolvidos com a área técnica reconhecerem a necessidade de uma abordagem mais disciplinada em relação ao software, entretanto, eles continuam a discutir a maneira como essa disciplina deve ser aplicada. Porém, muitos indivíduos e empresas desenvolvem software de forma desordenada, mesmo ao construírem sistemas dirigidos às mais avançadas tecnologias. Contudo, grande parte de profissionais e estudantes não estão cientes dos métodos modernos. porém, como conseqüência, a qualidade do software que produzem é afetada. Entretanto, além disso, a discussão e a controvérsia sobre a real natureza da abordagem de engenharia de software continuam. Por tanto, a engenharia de software é um estudo repleto de contrastes. A postura mudou, progressos foram feitos, porém, falta muito para essa disciplina atingir a maturidade.

Entretanto, segundo (KOSCIANSKI; SOARES, 2007) “a engenharia de software foi criada para resolver os problemas da crise do software, ou seja, para que os softwares produzidos tivessem qualidade a um preço e prazo razoáveis e que pudessem ser corretamente planejados”.

Contudo, (KOSCIANSKI; SOARES, 2007) ressaltam que: a qualidade de um software, como se pode ver, depende de se decidir o que significa qualidade! Não é um assunto que possa ser tratado com dogmas: “Não cometerás erros de programação”. Entretanto, em vez disso, é preciso adotar uma perspectiva técnica e considerar diversos fatores que afetam a construção do produto e que influenciem no julgamento dos usuários:

- Como primeiro fator o, tamanho e complexidade do software sendo construído;
- Como segundo fator o, número de pessoas envolvidas no projeto;
- Como terceiro fator, ferramentas utilizadas;
- Como quarto fator o, custos associados à existência de erros;
- Como quinto e ultimo fator o, custos associados à detecção e à remoção de erros.

Segundo (KOSCIANSKI; SOARES, 2007) o “conceito chamado de “zero - defeitos”; trata-se com certeza, de um conceito interessante e um ideal a ser buscado”. Contudo, “mas, de um ponto de vista de administração e de engenharia, é mais realístico se perguntar até que ponto pode-se evitar os erros em um dado projeto e, o que é decisivo, qual o custo e quais os lucros esperados”.

Para (ENGHOLM, 2010) a qualidade contempla uma série de objetivos da construção de software, objetivos esses, conhecidos como requisitos não-funcionais, tais como extensibilidade, com capacidade de manutenção, reutilização de códigos já desenvolvidos, desempenho, escalabilidade, usabilidade e confiabilidade nos dados apresentados pela aplicação. Entretanto, podemos presenciar uma série de problemas enfrentados por empresas que investem no desenvolvimento de sistemas sem a utilização de engenharia de software, porém, seja por desenvolvimento interno ou pela contratação de empresas que constroem sistemas.

Para (FULGENCIO, 2007) ad hoc significa “para este fim específico”, e segundo (ENGHOLM, 2010) para clarificar esses problemas, são apresentadas a seguir as conseqüências práticas de desenvolver softwares de modo ad hoc, sem utilização de processos definido, orientação a objetos e melhores práticas:

Primeiramente Softwares difíceis de se dar manutenção, tanto corretiva quanto evolutiva. Em segundo, Softwares difíceis de implementar alterações, tanto corretivas quanto evolutivas. Em terceiro a reutilização de código mal elaborado e sujeito a geração/propagação de erros em outras partes do sistema em desenvolvimento. Em quarto temos os sistemas com baixo desempenho e escalabilidade inadequada. Em uma quinta conseqüência temos baixa eficiência no desenvolvimento, com analistas desenvolvendo as mesmas funcionalidades diversas vezes. Em sexto temos a falta de confiança nos dados apresentados pelo sistema, fazendo com que usuários deixem de utilizar o sistema por não confiar nas informações apresentadas. E em sétimo a baixa qualidade de código.

Em contrapartida aos efeitos do desenvolvimento ad hoc, segundo (ENGHOLM, 2010) temos o maior desejo de todas as empresas que desenvolvem software: ter menores custos e tempo de desenvolvimento nos processos de implementação e manutenção de sistemas. Contudo, esse desejo faz com que muitas empresas prefiram não utilizar processos de engenharia de software, por este fim especifico, pensando na falsa economia de tempo e diminuição de custos. Entretanto, essa idéia é falsa, pois a não utilização de métodos adequados no desenvolvimento gera softwares de má qualidade, que trazem frustração aos usuários, custos adicionais relacionados à manutenção corretiva e problemas na utilização do sistema. Porém, desse modo, todo o tempo e dinheiro economizado no desenvolvimento será gasto em correções, trazendo prejuízos à imagem do projeto e ao próprio sistema.

Segundo (MARKS, 2008) “adhocracia é uma empresa ad hoc ou com características ad hoc. Vale dizer, são empresas que geram soluções, que podem ser destinadas para elas mesmas como para outras empresas ou outros setores”.

Para (SOMMERVILLE, 2011) quando falamos sobre a qualidade do software profissional, devemos levar em conta que o software é usado e alterado pelas pessoas, além de seus desenvolvedores. A qualidade, portanto, implica não apenas o que o software faz. Ao contrário, ela tem de incluir o comportamento do software enquanto ele está executando, bem como a estrutura e a organização dos programas do sistema e a documentação associada. Isso se reflete nos atributos de software chamados não funcionais ou de qualidade. Exemplos desses atributos são o tempo de resposta do software a uma consulta do usuário e a compreensão do código do programa.

Segundo (SOMMERVILLE, 2011) o fator mais significante em determinar quais técnicas e métodos de engenharia de software são mais importantes seja o tipo de aplicação a ser desenvolvida. Existem muitos tipos diferentes de aplicações existentes, inclui:

- Fator primeiro refere-se a aplicações stand-alone. Essas são as aplicações executadas em um computador local, como um PC. Todavia, elas contêm toda a funcionalidade necessária e não precisam estar conectadas a uma rede. Entretanto há exemplos de tais aplicações são aplicativos de escritório em um PC, programas CAD, software de manipulação de fotos etc.

- Fator segundo refere-se aplicações interativas baseadas em transações. São aplicações que executam em um computador remoto, acessadas pelos usuários a partir de seus computadores ou terminais. Contudo, certamente, aqui são incluídas aplicações Web como aplicações de comércio eletrônico em que você pode interagir com o sistema remoto para comprar produtos ou serviços. Porém, essa classe de aplicações também inclui sistemas corporativos, em que uma empresa fornece acesso a seus sistemas através de um navegador Web ou fornece outro tipo como um programa cliente especial e serviços baseados em nuvem, como é o caso de serviços de e-mail e compartilhamento de fotos. Portanto, aplicações interativas freqüentemente incorporam um grande armazenamento de dados, que é acessado e atualizado em cada transação.

- Fator terceiro refere-se aos sistemas de controle embutidos. São sistemas de controle que controlam e gerenciam dispositivos de hardware. Entretanto, numericamente, é provável que haja mais sistemas embutidos do que de qualquer outro tipo. Porém, exemplos de sistemas embutidos incluem softwares em telefone celular, softwares que controlam antitravamento de freios em um carro e software em um micro-ondas para controlar o processo de cozimento.

- Fator quarto refere-se aos sistemas de processamento de lotes. São sistemas corporativos projetados para processar dados em grandes lotes. Entretanto, eles processam grande número de entradas individuais para criar as saídas correspondentes. Exemplos de sistemas de lotes incluem sistemas periódicos de cobrança, como sistemas de cobrança telefônica, e sistemas de pagamentos de salário.

- Fator quinto refere-se aos sistemas de entretenimento. São sistemas cuja utilização principal é pessoal e cujo objetivo é entreter o usuário. Porém, a maioria desses sistemas é de jogos de diferentes tipos. Contudo, a qualidade de interação com o usuário é a característica particular mais importante dos sistemas de entretenimento.

- Fator sexto refere-se aos sistemas para modelagem e simulação. Entretanto, são sistemas que incluem vários objetos separados que interagem entre si, desenvolvidos por cientistas e engenheiros para modelar processos ou situações físicas. Porém, esses sistemas geralmente fazem uso intensivo de recursos computacionais e requerem sistemas paralelos de alto desempenho para executar.

- Fator sétimo refere-se aos sistemas de coleta de dados. Contudo, são sistemas que coletam dados de seu ambiente com um conjunto de sensores e enviam esses dados para outros sistemas para processamento. Por tanto, o software precisa interagir com sensores e freqüentemente é instalado em um ambiente hostil, por exemplo, dentro de uma máquina ou em um lugar remoto.

- Fator oitavo refere-se aos sistemas de sistemas. São sistemas compostos de uma série de outros sistemas de software. Entretanto, alguns deles podem ser produtos genéricos de software, como um programa de planilha eletrônica. Porém, outros sistemas do conjunto podem ser escritos especialmente para esse ambiente.

(KOSCIANSKI; SOARES, 2007) fazem a seguinte pergunta: “quais são os problemas enfrentados na construção e utilização de software?” Nas palavras de Koscianski e Soares, há uma pequena lista onde temos em:

- Primeiro os cronogramas não observados;
- Segundo os projetos com tantas dificuldades que são abandonados;
- Terceiro os módulos que não operam corretamente quando combinados;
- Quarto os programas que não fazem exatamente o que era esperado;
- Quinto os programa tão difícil de usar que são descartados;
- Sexto os programas que simplesmente param de funcionar.

Para (PÁDUA, 2012) geralmente a qualidade de um produto decorre diretamente da qualidade do processo utilizado na produção dele. Por tanto, note-se que importa aqui a qualidade do processo efetivamente utilizado, não do "processo oficial", contudo pode eventualmente estar descrito nos manuais da organização. Entretanto muitas vezes os processos oficiais não são seguidos na prática, por deficiência de ferramentas, por falta de qualificação das pessoas, ou entre outras, porque pressões de prazo levam os gerentes dos projetos a eliminar etapas relacionadas com controle da qualidade.

Segundo (PÁDUA, 2012) em um produto de software de má qualidade, muitos requisitos não são atendidos completamente. Entretanto, as deficiências de conformidade com os requisitos se manifestam de várias maneiras. Porém, em alguns casos, certas funções não são executadas corretamente sob certas condições, ou para certos valores de entradas. Contudo, em outros casos, o produto tem desempenho insuficiente, ou é difícil de usar.

Para (PÁDUA, 2012) cada requisito não atendido é um defeito. No mundo informático, criaram-se a usança de chamar de “bugs” os defeitos de software. Entretanto, assim, erros técnicos adquirem conotação menos negativa, que lembra simpáticos insetos de desenho animado. Por tanto o nome ajuda a esquecer que estes defeitos foram causados por erro de uma falível pessoa, e que cada defeito tem responsáveis bem precisos.

Note-se (PÁDUA, 2012) que defeitos incluem situações de falta de conformidade com requisitos explícitos, normativos e implícitos. Os defeitos associados a requisitos implícitos são os mais difíceis de tratar. Entretanto, eles

levam a desentendimentos sem solução entre o fornecedor e o cliente do produto. Portanto, além disto, como estes requisitos, por definição, não são documentados, é bastante provável que eles não tenham sido considerados no desenho do produto, o que tornará a correção dos defeitos particularmente trabalhosa.

Segundo (PÁDUA, 2012) em todas as fases do desenvolvimento de software as pessoas introduzem defeitos. Eles decorrem de limitações humanas: com erros lógicos, erros de interpretação, desconhecimento de técnicas, falta de atenção, ou falta de motivação. Contudo, em todo bom processo existem atividades de garantia da qualidade, tais como revisões, testes e auditorias. Estas atividades removem parte dos defeitos introduzidos.

Para (PÁDUA, 2012) quando atividades de controle da qualidade são cortadas, parte dos defeitos deixa de ser removida em um ponto do projeto. Defeitos que não são removidos precocemente acabam sendo detectados depois. Entretanto quanto mais tarde um defeito é corrigido, mais cara é a sua correção, por várias razões que serão discutidas posteriormente. Nota-se o pior caso acontece quando o defeito chega ao produto final. Neste caso, ele só será removido através de uma operação de manutenção. Porém, esta é a forma mais cara de remoção de defeitos. Em certos casos, como acontece em sistemas de missão crítica, defeitos de software podem trazer prejuízos irreparáveis.

2.7 DA EVOLUÇÃO DOS SISTEMAS E SUAS ROTINAS

Segundo (PRESSMAN; MAXIM, 2016) “programas mais antigos freqüentemente denominados softwares legado têm sido foco de contínua atenção e preocupação”, portanto, “um software legado é caracterizado pela longevidade e criticidade de negócio”. Entretanto, é importante ressaltar que uma das características é que trata-se de um “sistema antigo cujo processamento foi e ainda é importante para a empresa. Portanto, ele representa um legado. Porém, o importante é que ainda está em operação e desempenha funções vitais da empresa”. Enfatizando, sobretudo, “com o passar do tempo, a evolução dos sistemas, a concepção de programas capazes de gerenciar recursos ou prover serviços a vários outros programas, tornou-se largamente aceita”

Para (PRESSMAN, 2011) Infelizmente, algumas vezes, há uma característica adicional que pode estar presente em um software legado: baixa qualidade. Entretanto, os sistemas legados, algumas vezes, têm projetos não expansíveis,

código intrincado, documentação pobre ou inexistente, casos de testes e resultados que nunca foram arquivados, um histórico de modificações mal administrado. Contudo, a lista pode ser bem longa. Ainda assim, esses sistemas dão suporte a funções vitais de negócio e são indispensáveis para ele.

Para (PARETO, 2016) muitos sistemas foram desenvolvidos para grandes empresas: sistemas caros, complexos, com banco de dados da época, desenvolvidos para os computadores mainframes. Portanto, estes sistemas são extremamente estáveis e atendem até as demandas atuais. Porém, o que fazer com estes sistemas? O custo para desenvolver novos sistemas e abandonar os antigos é muito grande, então eles foram classificados como sistemas legados e continuam rodando. Contudo, estes sistemas fornecem serviços essenciais, mas têm difícil manutenção. Se procurarmos uma definição formal do termo sistema legada, certamente será difícil de encontrar. Por tanto se apresenta as seguintes características:

- Primeira característica refere-se ao sistema antigo cujo processamento foi e ainda é importante para a empresa. Ele representa um legado. O importante é que ainda está em operação e desempenha funções vitais da empresa.

- Segunda ou outra característica é que eles funcionam em hardwares obsoletos, principalmente nos mainframes, cujos componentes são extremamente caros e raros. Entretanto, estes sistemas utilizam linguagens como COBOL, banco de dados e formatos de arquivos obsoletos.

- Terceira característica refere-se a manutenção nestes sistemas é semelhante a um trabalho de restaurador histórico. De forma analógica, colocar a mão nestes sistemas é como colocar a mão em um vespeiro. Porém, geralmente, os profissionais que desenvolveram estes sistemas já se aposentaram ou estão em processo de aposentadoria.

- Quarta característica refere-se à documentação destes sistemas é falha, como na maioria dos sistemas. O problema é que identificar as regras de negócio implementadas não é uma tarefa simples.

Segundo (GUEDES, 2018) para facilitar a compreensão do sistema por quem tiver que mantê-lo, já que, em geral, a manutenção de um sistema é considerada uma tarefa ingrata pelos profissionais de desenvolvimento, todavia normalmente

exigir que estes despendam grandes esforços para compreender códigos escritos por outros cujos estilos de desenvolvimento são diferentes e que, via de regra, não se encontra mais na empresa.

Para (GUEDES, 2018) esse tipo de código é conhecido como “código alienígena” ou “software legado”. Contudo, o termo refere-se a códigos que não seguem as regras atuais de desenvolvimento da empresa, não foram modelados e, por conseguinte, têm pouca ou nenhuma documentação. Porém, além disso, nenhum dos profissionais da equipe atual trabalhou em seu projeto inicial e, para piorar, necessariamente o código já sofreu manutenções anteriores por outros profissionais que também não se encontram mais na empresa, sendo que cada um deles tinha um estilo de desenvolvimento diferente, ou seja, como se diz no meio de desenvolvimento, o código encontra-se “remendado.

2.8 DOS REQUISITOS NO DESENVOLVIMENTO DE ROTINAS

Nota-se (FRANCO, 2020) especificações, documentação e pessoas, são essenciais para o desenvolvimento de rotinas do sistema legado da organização. Segundo (KOSCIANSKI; SOARES, 2007) os requisitos foram especificados por uma pessoa: Portanto, logo, é preciso saber claramente do que ela precisa. No entanto, Mais do que isso, é preciso saber como cada pessoa envolvida no projeto, cliente, projetistas, gerentes, influi sobre os requisitos para conhecer com precisão o objetivo que se pretende alcançar.

As idéias da engenharia de software podem ser fundamentais e cruciais no sucesso do processo de desenvolvimento do software como um todo, segundo (SOMMERVILLE, 2011) “as idéias fundamentais da engenharia de software são universalmente aplicáveis para todos os tipos de desenvolvimento de sistemas. Esses fundamentos incluem processos de software gerenciados, confiança e proteção de software, engenharia de requisitos e reuso de software”. Nesse capítulo estamos abordando conceitos da engenharia de requisitos, por sua importância e relevância do tema.

Segundo (SOMMERVILLE, 2011) especificação de software ou engenharia de requisitos é o processo de compreensão e definição dos serviços requisitados do sistema e identificação de restrições relativas à operação e ao desenvolvimento do sistema. Entretanto, a engenharia de requisitos é um estágio particularmente crítico do processo de software, pois erros nessa fase inevitavelmente geram problemas no projeto e na implementação do sistema.

Para (SOMMERVILLE, 2011) o processo de engenharia de requisitos tem como objetivo produzir um documento de requisitos acordados que especifica um sistema que satisfaz os requisitos dos stakeholders. Requisitos são geralmente apresentados em dois níveis de detalhe. Os usuários finais e os clientes precisam de uma declaração de requisitos em alto nível; desenvolvedores de sistemas precisam de uma especificação mais detalhada do sistema.

Para (PRESSMAN, 2011) o amplo espectro de tarefas e técnicas que levam a um entendimento dos requisitos é denominado engenharia de requisitos. Entretanto, na perspectiva do processo de software, a engenharia de requisitos é uma ação de engenharia de software importante que se inicia durante a atividade de comunicação e continua na de modelagem. Portanto, ela deve ser adaptada às necessidades do processo, do projeto, do produto e das pessoas que estão realizando o trabalho. Todavia, a engenharia de requisitos constrói uma ponte para o projeto e para a construção.

Porém podem representar um acesso para os passos de um determinado projeto em construção. Segundo (PRESSMAN, 2011) a engenharia de requisitos constrói uma ponte para o projeto e para a construção. Entretanto, mas onde começa essa ponte? Alguém pode argumentar que ela começa na base dos interessados no projeto, ressaltando (por exemplo, gerentes, clientes, usuários finais), em que é definida a necessidade do negócio, são descritos cenários de usuários, delineados funções e recursos e identificadas restrições de projeto. Portanto, outros poderiam sugerir que ela se inicia com uma definição mais abrangente do sistema, em que o software é apenas um componente do domínio de sistema mais abrangente. Contudo porém, independentemente do ponto de partida, a jornada através da ponte nos leva bem à frente no projeto, permitindo que examinemos o contexto do trabalho de software a ser realizado, abrangendo assim, as necessidades específicas que o projeto e a construção devem atender, lembrando também as prioridades que orientam a ordem na qual o trabalho deve ser completado e as informações, funções e comportamentos que terão um impacto profundo no projeto resultante.

Para (PRESSMAN, 2011) a engenharia de requisitos fornece o mecanismo apropriado para entender aquilo que o cliente deseja, analisando as necessidades, avaliando a viabilidade, negociando uma solução razoável, todavia, especificando a solução sem ambigüidades validando a especificação e gerenciando as

necessidades à medida que são transformadas em um sistema operacional. Entretanto, ela abrange sete tarefas distintas: concepção, levantamento, elaboração, negociação, especificação, validação e gestão. Contudo, é importante notar que algumas delas ocorrem em paralelo e todas são adaptadas às necessidades do projeto.

Sobre o planejamento no levantamento de requisitos segundo (PRESSMAN, 2011) a atividade de planejamento (também denominada o jogo do planejamento) se inicia com a atividade de ouvir uma atividade de levantamento de requisitos que capacita os membros técnicos da equipe, leva-se a entender o ambiente de negócios do software e possibilita que se consiga ter uma percepção ampla sobre os resultados solicitados, fatores principais e funcionalidade.

Acredita-se ao realizar a tarefa de levantamento de requisito a parti de ouvir o cliente é fundamental no âmbito profissional. Segundo (PRESSMAN, 2011) a atividade de ‘ouvir’ conduz à criação de um conjunto de ‘histórias’ (também denominado histórias de usuários) que descreve o resultado, importante essas, as características e a funcionalidade requisitados para o software a ser construído. Cada história é escrita pelo cliente e é colocada em uma ficha. Entretanto, o cliente atribui um valor (uma prioridade) à história baseando-se no valor de negócio global do recurso ou função. Porém, os membros da equipe avaliam então cada história e atribui um custo medido em semanas de desenvolvimento a ela. Contudo, se a história requerer, por estimativa, mais do que três semanas de desenvolvimento, é solicitado ao cliente para dividir a história em histórias menores e a atribuição de valor e custo ocorre novamente. Todavia, é importante notar que podem ser escritas novas histórias a qualquer momento.

Para (VENTURA, 2016) “requisitos são o início de tudo, são a causa de “todo o resto” em projetos de software. Entendê-los corretamente, e materializar este entendimento no dia a dia, faz toda a diferença no sucesso dos projetos”. Entretanto, achar uma definição objetiva e exata para a palavra “requisito” é difícil. Portanto, a palavra possui alguns significados (conforme os dicionários), mas chega a ser um pouco abstrata. Contudo, mas basicamente, quando falamos de requisito, estamos falando de necessidade, exigência, desejo, solicitação. Porém, levando esta palavra para o contexto de um software, estamos falando de necessidades de um usuário, exigências do negócio, desejos da empresa, solicitação da empresa, embora tudo isso devendo ser realizado por um sistema, ou seja, o software deverá atender estas necessidades, exigências, desejos e solicitações, e materializar isso em um sistema.

Segundo (VENTURA, 2016) “em projetos de software é mais do que necessário haver apenas três tipos de requisito: Requisitos Funcionais, Requisitos Não Funcionais e Regras de Negócio”.

Sobre requisitos funcionais, para (VENTURA, 2016) é comum os profissionais de engenharia de software associem a idéia de um requisito funcional a uma tela, uma rotina, que no fim serão as funcionalidades de fato de um sistema. Entretanto, mas é necessário entender que uma funcionalidade não necessariamente realizará apenas um requisito funcional, podendo realizar vários requisitos funcionais cujo significa que em uma funcionalidade um ou mais requisitos funcionais podem ser atendidos, não necessariamente apenas um.

Sobre requisitos não funcionais, segundo (VENTURA, 2016) seria “como o próprio nome diz, é uma ‘não funcionalidade’, ou seja, trata-se de algo que não é uma funcionalidade, mas que precisa ser realizado para que o software atenda seu propósito”. Acredita-se sobretudo, as regras de negócio, “são restrições aplicadas a uma operação comercial de uma empresa, que precisam ser atendidas para que o negócio funcione da maneira esperada”.

Segundo (GUEDES, 2018) a fase de levantamento de requisitos deve identificar dois tipos de requisitos: em primeiro segue os funcionais e os não-funcionais. Entretanto, os requisitos funcionais correspondem ao que o cliente quer que o sistema realize, ou seja, as funcionalidades do software em si. Porém, já os requisitos não-funcionais correspondem às restrições, condições, consistências, validações que devem ser levadas a efeito sobre os requisitos funcionais. Portanto, exemplificando, em um sistema bancário deve ser oferecida a opção de abrir novas contas correntes, o que é um requisito funcional. Já determinar que somente pessoas maiores de idade possam abrir contas corrente é um requisito não-funcional.

Para (GUEDES, 2018) podem existir diversos tipos de requisitos não-funcionais, como de usabilidade, desempenho, confiabilidade, segurança ou interface. Entretanto, alguns requisitos não-funcionais identificam regras de negócio, ou seja, as políticas, normas e condições estabelecidas pela empresa que devem ser seguidas na execução de uma funcionalidade.

Continuando a abordagem sobre o levantamento e análise de requisitos,

segundo (GUEDES, 2018) uma das primeiras fases de um processo de desenvolvimento de software consiste no levantamento de requisitos. Entretanto, as outras etapas, sugeridas por muitos autores, são as seguintes: Análise de Requisitos, Projeto, que se constitui na principal fase da modelagem, Codificação, Testes e Implantação. Portanto, dependendo do método e processo adotado, essas etapas ganham, por vezes, nomenclaturas diferentes, podendo algumas delas ser condensadas em uma etapa única, ou uma etapa pode ser dividida em duas ou mais etapas. Todavia, se tomarmos como exemplo o Processo Unificado (Unified Process), um método de desenvolvimento de software verá que este se divide em quatro fases denominadas: Concepção, onde é feito o levantamento de requisitos; Elaboração, onde é feita a análise dos requisitos e o projeto do software; Construção, onde o software é implementado e testado; e Transição, onde o software será implantado. Entretanto, as fases de Elaboração e Construção ocorrem, sempre que possível, em ciclos iterativos, dessa forma, sempre que um ciclo é completado pela fase de Construção, concentra-se e volta-se à fase de Elaboração para tratar do ciclo seguinte, até todo o software ser finalizado.

Segundo (PÁDUA, 2012) o fluxo de requisitos reúne as atividades que visam obter o enunciado completo, claro e preciso dos requisitos de um produto de software. Entretanto, estes requisitos devem ser levantados pela equipe do projeto, em conjunto com representantes do cliente, usuários chaves e outros especialistas da área de aplicação. Portanto, o conjunto de técnicas empregadas para levantar, detalhar, documentar e validar os requisitos de um produto forma a Engenharia de Requisitos. Porém, o resultado principal do fluxo dos requisitos é um documento de especificação de requisitos de software.

Para (PÁDUA, 2012) projetos de produtos mais complexos geralmente precisam de maior investimento em engenharia de requisitos que projetos de produtos mais simples. Entretanto, a Engenharia de Requisitos é também mais complexa no caso de produtos novos. Porém, quando um projeto visa desenvolver uma nova versão de um produto existente, a experiência dos usuários com as versões anteriores permite identificar de forma rápida e clara as necessidades prioritárias. Portanto, no caso de um novo produto, é mais difícil para os usuários identificar quais as características de maior valor, e é mais difícil para os desenvolvedores entender claramente o que os usuários desejam.

Segundo (PÁDUA, 2012) uma boa engenharia de requisitos é um passo essencial para o desenvolvimento de um bom produto, em qualquer caso.

Entretanto, este capítulo descreve de forma detalhada as atividades do fluxo de Requisitos, assim como algumas das técnicas mais importantes para a obtenção de requisitos de alta qualidade. Portanto, para garantir ainda mais a qualidade, os requisitos devem ser submetidos aos procedimentos de controle das fases do processo, e devem ser verificados através das atividades de análise.

Nota-se (PÁDUA, 2012) os requisitos de alta qualidade são claros, completos, sem ambigüidade, implementáveis, consistentes e testáveis. Portanto, os requisitos que não apresentem estas qualidades são problemáticos, portanto, “eles devem ser revistos e renegociados com os clientes e usuários.

Para (PÁDUA, 2012) os requisitos estão contidos nos artefatos enumerados. Entretanto, a proposta de especificação do Software contém uma visão preliminar dos requisitos, que será usada apenas para iniciar o fluxo de Requisitos, e não será mantida dentro das linhas de base do projeto. Portanto, os demais artefatos fazem parte das linhas de base. Contudo, o enunciado detalhado dos requisitos estará contido na Especificação dos Requisitos do Software. Entretanto, o modelo dos casos de uso, parte da descrição dos requisitos funcionais, estará contido no Modelo de Análise do Software. Porém, o cadastro dos requisitos do Software é a base de dados que contém uma lista sumária de todos os requisitos e dos relacionamentos destes com itens derivados, gerados pelos demais fluxos do processo.

Segundo (PÁDUA, 2012) os requisitos de um produto podem alterar-se ao longo de seu desenvolvimento por diversos motivos, dos quais descartamos em:

- Primeiro a descoberta de defeitos e inadequações nos requisitos originais;
- Segundo a falta de detalhes suficientes nos requisitos originais;
- Terceiro a alterações incontornáveis no contexto do projeto (por exemplo, mudanças de legislação).

Para (PÁDUA, 2012) mesmo reconhecendo este fato, todo o esforço deve ser feito para que a especificação dos requisitos do software seja tão completa quanto possível. Portanto, no caso de alterações serem indispensáveis, elas devem obedecer aos procedimentos de gestão de requisitos de software.

Nota-se (PÁDUA, 2012) que segundo o paradigma SW-CMM, uma organização considerada madura na gestão de requisitos de software deve atingir as seguintes metas:

• Primeiramente, os requisitos de software são controlados para estabelecer uma base para as atividades gerenciais e de engenharia de software, portanto, dentro de um projeto.

• Em segundo, os planos, resultados, produtos e atividades de software são mantidos consistentes com os requisitos de software.

Segundo (PÁDUA, 2012) para servir de base a um produto de boa qualidade, a própria especificação de requisitos deve satisfazer uma série de características de qualidade. As características mais importantes são as seguintes:

• Correta - Caracterizada por todo requisito presente se realmente é um requisito do produto a ser construído.

• Precisa - Caracterizada por todo requisito presente possui apenas uma única interpretação, aceita tanto pelos desenvolvedores quanto pelos usuários chaves.

• Completa - Caracterizada por refletir todas as decisões das especificações que foram tomadas.

• Consistente - Caracterizada por verificar se não há conflitos entre nenhum dos subconjuntos de requisitos presentes.

• Priorizada - Caracterizada por verificar se cada requisito é classificado de acordo com a sua importância, estabilidade e complexidade.

• Verificável - Caracterizada por verificar se todos os seus requisitos são verificáveis.

• Modificável - Caracterizada por verificar se sua estrutura e estilo permitem a mudança de qualquer requisito, de forma fácil, completa e consistente.

• Rastreável - Caracterizada por permitir a fácil determinação dos antecedentes e conseqüências de todos os requisitos.

Existem importantes características de qualidade que são especificações fundamentais dos requisitos essenciais para o sucesso do projeto do software. (PÁDUA, 2012), comenta cada uma das características de qualidade conforme a seguir:

Em se tratando de correção, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é correta se todo requisito presente nela realmente é um requisito do produto a ser construído. Entretanto, não existe ferramenta que garanta a correção de uma Especificação dos Requisitos. Portanto, para verificá-la, deve-se checar a coerência da Especificação dos Requisitos do Software com

outros documentos da aplicação, tais como a Proposta de Especificação do Software, a Especificação dos Requisitos do Sistema e outros padrões referentes à área de aplicação. Porém, deve-se ainda solicitar a aprovação formal da Especificação dos Requisitos do Software por parte do cliente, sem a qual o projeto não poderá prosseguir.

Em se tratando de precisão, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é precisa se todo requisito presente possuir apenas uma única interpretação, aceita tanto pelos desenvolvedores quanto pelos usuários chaves. Entretanto, em particular, uma Especificação dos Requisitos deve ser compreensível para todo o seu público alvo, e deve ser suficiente para o desenho dos testes de aceitação. Portanto, recomenda-se a inclusão no glossário da Especificação dos Requisitos de todos os termos contidos no documento que possam causar ambiguidades em sua interpretação. Contudo, os seguintes meios devem ser usados para garantir maior precisão da Especificação dos Requisitos:

• os meios de revisões técnicas;
• os meios de uso de notações e ferramentas de análise, orientadas a objetos.

Em se tratando de completeza, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é completa se reflete todas as decisões de especificação que foram tomadas, não contendo cláusulas de pendências. Contudo, uma Especificação dos Requisitos completa deve:

• contemplar e conter todos os requisitos significativos relativos à funcionalidade, desempenho, restrições de desenho, atributos e interfaces externas;
• contemplar e definir as respostas do software para todas as entradas possíveis, válidas e inválidas, em todas as situações possíveis;
• contemplar e conter um glossário de todos os termos técnicos e unidades de medida, assim como referências completas a todos os diagramas, figuras e tabelas.

Em se tratando de consistência, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é consistente se não há conflitos entre nenhum dos subconjuntos de requisitos presentes. Entretanto, existem três tipos comuns de conflitos entre requisitos:

• conflitos comuns entre características de objetos do mundo real - por exemplo, formatos de relatórios ou cores de sinalização;

• conflito comum lógico ou temporal entre ações - por exemplo, um requisito diz que a ação A deve ser realizada antes da ação B, e outro diz o contrário;

• conflitos comuns de uso de termos diferentes para designar o mesmo objeto do mundo real - por exemplo, “lembrete” versus “mensagem”.

Em se tratando de Priorização, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é priorizada se cada requisito é classificado de acordo com a respectiva importância e estabilidade. Entretanto, a estabilidade estima a probabilidade de que o requisito venha a ser alterado no decorrer do projeto, com base na experiência de projetos correlatos. Portanto, a priorização classifica o requisito de acordo com um dos seguintes graus:

• Priorização de requisito essencial – requisito sem cujo atendimento o produto é inaceitável;

• `Priorização de requisito desejável – requisito cujo atendimento aumenta o valor do produto, mas cuja ausência pode ser relevada em caso de necessidade (por exemplo, de prazo);

• Priorização de requisito opcional – requisito a ser cumprido se houver disponibilidade de prazo e orçamento, depois de atendidos os demais requisitos.

Em se tratando de Verificabilidade, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é verificável se todos os seus requisitos são verificáveis. Entretanto, um requisito é verificável se existir um processo finito, com custo compensador, que possa ser executado por uma pessoa ou máquina, e que mostre a conformidade do produto final com o requisito. Portanto, em geral requisitos ambíguos não são verificáveis, assim como requisitos definidos em termos qualitativos, ou contrários a fatos técnicos e científicos.

Em se tratando de Modificabilidade, segundo (PÁDUA, 2012), se trata de uma Especificação dos Requisitos é modificável se sua estrutura e estilo permitirem a mudança de qualquer requisito, de forma fácil, completa e consistente. Entretanto, a modificabilidade geralmente requer:

• Nível de organização coerente, com índices e referências cruzadas;
• Nível de ausência de redundância entre requisitos;
• Nível de definição separada de cada requisito.

Em se tratando de Rastreabilidade, segundo (PÁDUA, 2012), se trata de uma

Especificação dos Requisitos é rastreável se permite a fácil determinação dos antecedentes e conseqüências de todos os Requisitos. Entretanto, dois tipos de rastreabilidade devem ser observados.

• Tipos de rastreabilidade para trás - deve ser possível localizar a origem de cada requisito. Entretanto, deve-se sempre saber porque existe cada requisito, e quem ou o que o originou. Portanto, isto é importante para que se possa avaliar o impacto da mudança daquele requisito, e dirimir dúvidas de interpretação.

• Tipos de rastreabilidade para frente - deve ser possível localizar quais os resultados do desenvolvimento que serão afetados por cada requisito. Entretanto, isto é importante para garantir que os itens de análise, desenho, código e testes cubram todos os requisitos, e para localizar os itens que serão afetados por uma mudança nos requisitos.

2.8.1 **Da documentação**

Segundo (PARETO, 2016), o analista de sistema “tem contato direto com os clientes, para verificar suas necessidades, requisitos e ou problemas que os usuários possam ter. Faz uma análise inicial, depois disso emite uma documentação, para que o sistema possa ser construído”. A documentação para (SOMMERVILLE, 2011), é uma das “atividades que dão apoio ao processo”, e também o “gerenciamento de configuração de software”. Segundo (PRESSMAN, 2011), para que “faça uma documentação que seja autodocumentável”. Ressalta também que se trata de um mito disser, que “ a engenharia de software nos fará criar documentação volumosa e desnecessária e, invariavelmente, irá nos retardar”. Contudo, ressalta ainda que “erros na documentação podem ser tão devastadores para a aceitação dos programas quanto os erros nos dados ou no código-fonte”.

Segundo (GAMMA et al., 2000) os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Entretanto, expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas. Portanto, os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização. Contudo, os padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Porém, em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.

Segundo (GUEDES, 2018) por mais simples que seja, todo e qualquer sistema deve ser modelado antes de se iniciar sua implementação, todavia, entre outras coisas, porque os sistemas de informação freqüentemente costumam ter a propriedade de “crescer”, porém, isto é, aumentar em tamanho, complexidade e abrangência”. Entretanto, muitos profissionais costumam afirmar que sistemas de informação são “vivos”, porque nunca estão completamente finalizados. Porém, na verdade, o termo correto seria “dinâmico”, pois normalmente os sistemas de informação estão em constante mudança”. Contudo, “tais mudanças são devidas a diversos fatores, como, por exemplo,”:

• Mudanças por parte dos clientes desejam constantemente modificações ou melhorias no sistema.

• Mudanças por parte do mercado estão sempre mudando, o que força a adoção de novas estratégias por parte das empresas e, conseqüentemente, de seus sistemas.

• Mudanças por parte do governo, seguidamente promulga novas leis e cria novos impostos e alíquotas ou, ainda, modifica as leis, os impostos e alíquotas já existentes, o que acarreta a manutenção no software.

Dessa forma, segundo (GUEDES, 2018) assim, um sistema de informação precisa ter uma documentação extremamente detalhada, precisa e atualizada para que possa ser mantido com facilidade, rapidez e correção, porém sem produzir novos erros ao corrigir os antigos. Entretanto, modelar um sistema é uma forma bastante eficiente de documentá-lo, mas a modelagem não serve apenas para isso: a documentação é apenas uma das vantagens fornecidas pela modelagem.

Para (GUEDES, 2018) ao perguntar, qual a real necessidade de se modelar um software? Ressalta que muitos “profissionais” podem afirmar que conseguem determinar todas as necessidades de um sistema de informação de cabeça, e que sempre trabalharam assim. Entretanto, mas a questão é muito mais ampla, envolvendo fatores extremamente complexos, como levantamento e análise de requisitos, sobretudo, prototipação, tamanho do projeto, complexidade, prazos, custos, documentação, manutenção e reusabilidade, entre outros.

Contudo, segundo (GUEDES, 2018) uma modelagem correta aliada a uma documentação completa e atualizada de um sistema de informação torna mais rápido o processo de manutenção e impede que erros sejam cometidos, entretanto, já que é muito comum que, depois de manter uma rotina ou função de um software,

outras rotinas ou, sobretudo funções do sistema que antes funcionavam perfeitamente passem a apresentar erros ou simplesmente deixem de funcionar. Tais erros são conhecidos como “efeitos colaterais” da manutenção.

Para (GUEDES, 2018), além disso, qualquer manutenção a ser realizada em um sistema deve ser também modelada e documentada, para não desatualizar a documentação do sistema e prejudicar futuras manutenções, a uma preocupação já que muitas vezes uma documentação desatualizada pode ser mais prejudicial à manutenção do sistema do que nenhuma documentação.

Segundo (GUEDES, 2018) uma empresa ou setor de desenvolvimento de software necessita de um registro detalhado de cada um de seus sistemas de informação antes desenvolvidos para poder determinar, entre outros, fatores como:

• Os fatores que determina a média de manutenções que um sistema sofre normalmente dentro de um determinado período de tempo.
• Os fatores que determina a qual a média de custo de modelagem.
• Os fatores que determina a qual a média de custo de desenvolvimento.
• Os fatores que determina a qual a média de tempo despendido até a finalização do projeto.
• Os fatores que determina quantos profissionais são necessários envolver normalmente em um projeto.

Para (GUEDES, 2018) essas informações são computadas nos orçamentos de desenvolvimento de novos softwares e são de grande auxílio no momento de determinar prazos e custos mais próximos da realidade. Entretanto, além disso, a documentação pode ser muito útil em outra área: a Reusabilidade. Portanto, um dos grandes desejos e muitas vezes necessidades dos clientes é que o software esteja concluído o mais rápido possível. Contudo, uma das formas de agilizar é a reutilização de rotinas, funções e algoritmos previamente desenvolvidos em outros sistemas. Sobretudo nesse caso, a documentação correta do sistema pode auxiliar a sanar questões como:

• Primeiro, onde as rotinas se encontram?
• Segundo, para que foram utilizadas?
• Terceiro, em que projetos estão documentados?
• Quarto, elas são adequadas ao software atualmente em desenvolvimento?
• Quinto qual o nível necessário de adaptação destas rotinas para que possam ser utilizadas na construção do sistema atual?

Em se tratando de padrões de projetos, segundo (GAMMA et al., 2000) os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Entretanto, expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas. Contudo, os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização. Porém, os padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Sobretudo em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.

2.8.2 **Da especificação**

Para (SOMMERVILLE, 2011), “os processos de especificação, projeto e implementação são intercalados. Não há especificação detalhada do sistema, e a documentação do projeto é minimizada ou gerada automaticamente pelo ambiente de programação usado para implementar o sistema”. Entretanto, o usuário tem a sua participação em se tratando de requisitos, “o documento de requisitos do usuário apenas define as características mais importantes do sistema”. Segundo (PÁDUA, 2012), “Especificação dos Requisitos do Software” é um “documento que descreve, de forma detalhada, o conjunto de requisitos especificados para um produto de software”. Nota-se que para (KOSCIANSKI; SOARES, 2007) “um dos fatores que exerce influência negativa sobre a qualidade de um projeto é a complexidade, que está associada a uma característica bastante simples: o tamanho das especificações”.

Segundo (SOMMERVILLE, 2011) um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software. Entretanto, essas atividades podem envolver o desenvolvimento de software a partir do zero em uma linguagem padrão de programação como Java ou C. Porém, no entanto, aplicações de negócios não são necessariamente desenvolvidas dessa forma. Sobretudo atualmente, novos softwares de negócios são desenvolvidos por meio da extensão e modificação de sistemas existentes ou por meio da configuração e integração de prateleira ou componentes do sistema.

Para (SOMMERVILLE, 2011) existem muitos processos de software diferentes, mas todos devem incluir quatro atividades fundamentais para a

engenharia de software. São ela:

1. Atividades como especificação de software. A funcionalidade do software e as restrições a seu funcionamento devem ser definidas.
2. Atividades como projeto e implementação de software. O software deve ser produzido para atender às especificações.
3. Atividades como validação de software. O software deve ser validado para garantir que atenda às demandas do cliente.
4. Atividades como evolução de software. O software deve evoluir para atender às necessidades de mudança dos clientes.

Continuando e ressaltando, segundo (SOMMERVILLE, 2011), “especificação de software ou engenharia de requisitos é o processo de compreensão e definição dos serviços requisitados do sistema e identificação de restrições relativas à operação e ao desenvolvimento do sistema”. Outra observação citada é: “a engenharia de requisitos é um estágio particularmente crítico do processo de software, pois erros nessa fase inevitavelmente geram problemas no projeto e na implementação do sistema”. De acordo com (KOSCIANSKI; SOARES, 2007), “os desenvolvedores do produto podem se encontrar face a um duplo problema, resolver ou, pelo menos, minimizar o problema organizacional do cliente que contrata o desenvolvimento do software e, depois, obter uma especificação coerente que atenda aos interessados”.

Para (SOMMERVILLE, 2011) os usuários finais e os clientes precisam de uma declaração de requisitos em alto nível; desenvolvedores de sistemas precisam de uma especificação mais detalhada do sistema. Entretanto, existem quatro atividades principais do processo de engenharia de requisitos:

- Atividades como estudo de viabilidade. É feita uma estimativa acerca da possibilidade de se satisfazerem as necessidades do usuário identificado usando-se tecnologias atuais de software e hardware. Entretanto, o estudo considera se o sistema proposto será rentável a partir de um ponto de vista de negócio e se ele pode ser desenvolvido no âmbito das atuais restrições orçamentais. Portanto, um estudo de viabilidade deve ser relativamente barato e rápido. O resultado deve informar a decisão de avançar ou não, com uma análise mais detalhada.

- Atividades como elicitação e análise de requisitos. Esse é o processo de

derivação dos requisitos do sistema por meio da observação dos sistemas existentes, além de discussões com os potenciais usuários e compradores, porém a análise de tarefas, entre outras etapas. Entretanto, essa parte do processo pode envolver o desenvolvimento de um ou mais modelos de sistemas e protótipos, os quais nos ajudam a entender o sistema a ser especificado.

- Atividades como especificação de requisitos. É a atividade de traduzir as informações obtidas durante a atividade de análise em um documento que defina um conjunto de requisitos. Entretanto, dois tipos de requisitos podem ser incluídos nesse documento. Portanto, requisitos do usuário são declarações abstratas dos requisitos do sistema para o cliente e usuário final do sistema; requisitos de sistema é uma descrição mais detalhada da funcionalidade a ser provida.

- Atividades como a validação de requisitos. Essa atividade verifica os requisitos quanto a realismo, consistência e completude. Durante esse processo, os erros no documento de requisitos são inevitavelmente descobertos. Sobretudo em seguida, o documento deve ser modificado para correção desses problemas.

Para (FREITAS, 2015) ao final do processo de engenharia de software é obtida a especificação de um produto de software. Entretanto, esta especificação deve garantir que as necessidades e expectativas do cliente e dos usuários do sistema sejam atendidas. Portanto, esta garantia é dada pela engenharia de requisitos que, se bem conduzida, fornecerá subsídios, na forma de documentos de requisitos, para garantir que a especificação está em acordo com o que se espera do produto final de software. Contudo, o resultado da engenharia de requisitos é um ou mais documentos de requisitos que garantam que a especificação do sistema satisfaz as necessidades do cliente e dos usuários. Porém, a garantia de que a especificação atenda tanto cliente quanto usuários são realizados por meio de várias atividades já relacionadas.

Em se tratando de padrões de projetos, segundo (GAMMA et al., 2000) os padrões de projeto podem melhorar a documentação e a manutenção de sistemas ao fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Entretanto, em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.

Para (PÁDUA, 2012) a especificação dos requisitos do software é o documento oficial de descrição dos requisitos de um projeto de software. Entretanto, ela pode se referir a um produto indivisível de software, ou a um conjunto de componentes de software, sobretudo forma um produto quando usados em conjunto (por exemplo, um módulo cliente e um módulo servidor).

“Segundo (PÁDUA, 2012) as características que devem estar contidas na especificação dos requisitos do software incluem”:

- Especificações no quesito funcionalidade: O que o software deverá fazer?
- Especificações no quesito interfaces externas: Como o software interage com as pessoas, com o hardware do sistema, com outros sistemas e com outros produtos?
- Especificações no quesito desempenho: Qual a velocidade de processamento, o tempo de resposta e outros parâmetros de desempenho requeridos pela natureza da aplicação?
- Especificações no quesito outros atributos: Quais as considerações sobre portabilidade, manutenibilidade e confiabilidade que devem ser observadas?
- Especificações no quesito restrições impostas pela aplicação: Existem padrões e outros limites a serem obedecidos, como linguagem de implementação, ambientes de operação, limites de recursos etc.

Segundo (PÁDUA, 2012) a especificação dos requisitos do software deve ser escrita por membros da equipe de desenvolvimento de um projeto, com a participação obrigatória de um ou mais usuários chaves do produto em pauta. Entretanto, o usuário chave é aquele que é indicado pelo cliente como pessoa capacitada a definir requisitos do produto, sobretudo normalmente, os usuários chaves são escolhidos entre profissionais experientes das diversas áreas que usarão o produto. Portanto, estes usuários chaves devem ser devidamente informados e treinados sobre as técnicas e notações que serão utilizadas no fluxo de requisitos.

Para (PÁDUA, 2012) geralmente, nem desenvolvedores nem clientes ou usuários são qualificados para escrever por si sós a especificação dos requisitos do software, por que:

- Na grande maioria os clientes nem sempre entendem os processos de

desenvolvimento de software em grau suficiente para produzir uma especificação de requisitos de implementação viável;

- Na grande maioria os desenvolvedores nem sempre entendem a área de aplicação de forma suficiente para produzir uma especificação de requisitos satisfatória.

Segundo (PÁDUA, 2012) os usuários chaves devem ser conscientizados do papel essencial que desempenham na especificação dos requisitos do software. Entretanto, deve-se, também, comunicar-lhes o papel que terão no restante do projeto, tal como no desenho das interfaces de usuário (inclusive estudos de usabilidade), sobretudo, revisões técnicas e de apresentação, avaliação das liberações, testes de aceitação e todos os procedimentos de implantação.

Para (PÁDUA, 2012) o mais completo destes documentos é a especificação dos requisitos do sistema. Entretanto, esta especificação definirá os requisitos aplicáveis ao sistema como um todo. Portanto, estes requisitos podem ser repassados aos componentes de software, ou realizados por outros componentes. Sobretudo, além disto, o desenho do sistema definirá as interfaces entre os componentes de software e os demais componentes. Contudo, estas interfaces podem resultar em requisitos adicionais de software. Os requisitos dos componentes do software não poderão entrar em conflito com os requisitos do sistema total.

Segundo (PÁDUA, 2012) quando o software fizer parte de um sistema maior que está sendo especificados de forma concorrente, sobretudo os requisitos de todo o sistema e de seus componentes separados passam a ser definidos em conjunto pelas diversas equipes do sistema, e negociados entre elas. Entretanto, exemplos de equipes com as quais a equipe de especificação de software pode ter de interagir incluem os desenvolvedores de hardware, portanto redes e bancos de dados e os especialistas da área de aplicação, além de pessoal de marketing e de áreas administrativas e financeiras.

Nota-se (PÁDUA, 2012) a equipe do projeto de software deve atuar juntamente com esses demais grupos, e com clientes e usuários chaves, na definição dos requisitos de nível de sistema. Entretanto, ela deve sempre indicar para os demais participantes do levantamento de requisitos de sistema se os requisitos que se pretende implementar por meio de software são viáveis. Contudo, todo requisito de sistema que tenha impacto no desenvolvimento de software deve ser aprovado pelo gerente do projeto de software.

Segundo (PÁDUA, 2012) durante o desenvolvimento dos requisitos de sistema, os grupos participantes devem definir quais características dos requisitos são críticas, do ponto de vista dos clientes e usuários. Entretanto, devem também estabelecer critérios de aprovação para cada componente do sistema que um grupo deva fornecer a outros grupos.

Porém, para (PÁDUA, 2012) normalmente, a especificação dos requisitos do software não deve incluir decisões de desenho e implementação, nem aspectos gerenciais de projeto. Entretanto, uma exceção é o caso em que estes aspectos são restrições definidas pelo cliente. Por exemplo, este pode definir que serão usadas determinadas linguagens de programação, sobretudo determinados componentes ou determinadas plataformas de bancos de dados. Por isto, segundo (PÁDUA, 2012) a especificação dos requisitos do software deverá satisfazer os seguintes critérios:

- Satisfazer a definição completa e corretamente todos os requisitos do produto do software. Porém, requisitos podem existir em virtude da natureza do problema a ser resolvido, ou em virtude de outras características específicas do projeto.
- Satisfazer em não descrever qualquer detalhe de desenho ou de implementação. Porém, estes devem ser descritos nos modelos e documentos produzidos pelos respectivos fluxos.
- Satisfazer em não descrever aspectos gerenciais do projeto, como custos e prazos. Entretanto, estes devem ser especificadas em outros documentos, tais como o Plano de Desenvolvimento do Software ou o Plano da Qualidade do Software.

Segundo (PÁDUA, 2012) normalmente, os seguintes itens são considerados como parte do desenho, e não devem fazer parte da especificação dos requisitos do software:

- Item um, partição do produto em módulos;
- Item dois, alocação de funções aos módulos;
- Item três, fluxo de informação entre módulos;
- Item quatro, estruturas internas de dados.

Para (PÁDUA, 2012) os requisitos a seguir são considerados requisitos gerenciais do projeto, e não devem ser incluídos na especificação dos requisitos do software:

- Gerenciais de custo;
- Gerenciais de cronograma de entregas;
- Gerenciais de relatórios requeridos;
- Gerenciais de métodos requeridos de desenvolvimento;
- Gerenciais de procedimentos de controle da qualidade;
- Gerenciais de “critérios de verificação e validação”.

2.9 DA ORGANIZAÇÃO DOS PROCESSOS DAS ROTINAS

Para (PARETO, 2016), “com a crescente importância da TIC para os negócios das organizações”, podemos disser que é estratégico? De certa forma, “explora e aperfeiçoa as tecnologias e”, sobretudo “a conexão entre a ferramenta e a estratégia de negócios, ou seja,”, por isso, “usa a tecnologia mais apropriada de forma a apoiar a estratégia de negócios da organização, necessitando, para tanto, de uma alta capacidade de gestão, conhecimento e visão estratégica do negócio”. É estratégico centralizar e organizar rotinas? Continuando, “coordena a implementação desse plano, observando cronogramas, prioridades e orçamentos aprovados”.

Segundo (SOMMERVILLE, 2011), as organizações “precisam decidir como obter o melhor retorno de seus investimentos, o que envolve fazer uma avaliação realista de seus sistemas legados e, em seguida, decidir sobre a estratégia mais adequada para a evolução desses sistemas”. O autor continua:

Reestruturar o sistema para melhorar sua manutenibilidade. Portanto, essa opção deve ser escolhida quando a qualidade do sistema foi degradada pelas mudanças, e novas mudanças para o novo sistema ainda estão sendo propostas. Entretanto, esse processo pode incluir o desenvolvimento de novos componentes de interface, para que o sistema original possa trabalhar com outros sistemas mais novos.

Para (SOMMERVILLE, 2011) os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Entretanto, não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software. Portanto, os processos têm evoluído de maneira a tirarem melhor proveito das capacidades das pessoas em uma organização, bem como das características específicas do sistema em desenvolvimento. Contudo, para alguns sistemas, como

sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado, sobretudo para sistemas de negócios, com requisitos que se alteram rapidamente, provavelmente será mais eficaz um processo menos formal e mais flexível.

Ainda segundo (SOMMERVILLE, 2011) embora não exista um processo ideal de software, há espaço, em muitas organizações, para melhorias no processo de software. Entretanto, os processos podem incluir técnicas ultrapassadas ou não aproveitar as melhores práticas de engenharia de software da indústria. Portanto, de fato, muitas empresas ainda não se aproveitam dos métodos da engenharia de software em seu desenvolvimento de software.

Para (SOMMERVILLE, 2011) em organizações nas quais a diversidade de processos de software é reduzida, os processos de software podem ser melhorados pela padronização. Entretanto, isso possibilita uma melhor comunicação, além de redução no período de treinamento, e torna mais econômico o apoio ao processo automatizado. Contudo, a padronização também é um importante primeiro passo na introdução de novos métodos e técnicas de engenharia de software, assim como as boas práticas de engenharia de software.

Segundo (GAMMA et al., 2000) estudos de programadores especialistas em linguagens convencionais mostraram que o conhecimento e a experiência não são organizados simplesmente em torno da sintaxe, portanto, mas sim em torno de estruturas conceituais maiores, tais como algoritmos, estruturas de dados e termos, e planos para atingir um objetivo específico. Porém, os projetistas provavelmente não pensam sobre a notação que usam para registrar decisões de projeto, enquanto estão tentando resolver a situação atual do projeto com planos, algoritmos, sobretudo em estrutura de dados e termos que aprenderam no passado.

Para (GAMMA et al., 2000) os cientistas da computação nomeiam e catalogam algoritmos e estruturas de dados, porém, raramente nomeamos outros tipos de padrões. Entretanto, os padrões de projeto fornecem um vocabulário comum para comunicar, documentar e explorar alternativas de projeto. Portanto, os padrões de projeto tornam um sistema menos complexo ao permitir falar sobre ele em um nível de abstração mais alto do que aquele de uma notação de projeto ou uma linguagem de programação. Contudo, os padrões de projeto elevam o nível no qual você projeta e discute o projeto com seus colegas.

Segundo (PÁDUA, 2012) um processo é definido quando tem documentação que detalha: o que é feito (produto), quando (passos), por quem (agentes), as coisas que usa (insumos) e as coisas que produz (resultados). Entretanto, processos podem ser definidos com mais ou menos detalhes, como acontece com qualquer receita. Portanto, os passos de um processo podem ter ordenação apenas parcial, o que pode permitir paralelismo entre alguns passos.

Para (PÁDUA, 2012) grandes mudanças de processo têm de começar do topo. Toda mudança requer liderança baseada em uma visão estratégica. Entretanto, esta visão deve ser encampada por patrocinadores com suficiente força dentro da organização. Porém, eles podem sustentar custos que são certos e de curto prazo, a troco de benefícios incertos de médio e longo prazo. Contudo, eles podem ter uma visão de sobrevivência da organização, mais abrangente que os resultados deste semestre ou deste ano. sobretudo, eles têm um escopo de interesse mais amplo que o de cada projeto.

Nota-se (PÁDUA, 2012) problemas de processo, são de responsabilidade dos gerentes dos projetos. Entretanto, só através da melhoria de processos é possível evitar a repetição de erros e incorporar as lições dos projetos passados. Contudo, o gerente de projeto que não tem como prioridade a melhoria dos processos está condenado a apagar incêndios eternamente. Sobretudo, é papel do gerente de projeto propor a sua equipe metas desafiadoras, em termos de custos, prazos e qualidade dos resultados. Entretanto, por outro lado, estas metas devem ser claramente viáveis, sob pena de não serem realmente levadas a sério. Portanto, uma vez propostas as metas de processo, elas devem ser cobradas, insistindo-se em criar uma cultura de excelência.

Segundo (PÁDUA, 2012) conseqüentemente ou naturalmente, erros e problemas acontecerão. Entretanto, o gerente de projeto deve então procurar as deficiências de processo que causaram os problemas. Portanto, gerentes que dão prioridade à caça de culpados estão, na realidade, convidando seus subordinados a esconder problemas. Contudo, gerentes que são os primeiros a quebrar as regras estão deixando bem claro que estas regras não são realmente para valer.

Para (PÁDUA, 2012) todos os que serão afetados pelas mudanças de processos têm de ser envolvidos. O custo deste envolvimento é sempre baixo comparado com os prejuízos que podem, sobretudo ser causados por quem teme a mudança, não a leva a sério, ou não a entende. Processos imaturos são baseados

na improvisação individual. Entretanto, eles fomentam nas pessoas mal informadas sensações de liberdade, criatividade e até poder, que na realidade são ilusórias, porque pagas com riscos, incerteza e desperdício. Porém, processos maduros permitem a ação mais estruturada, eficiente e coletiva. Reduzindo a incerteza e o estresse, eles contribuem para melhor qualidade de vida no trabalho.

Segundo (PÁDUA, 2012) a mudança eficaz requer conhecimento dos processos atuais. Entretanto, não adianta ter um mapa quando não sabemos onde estamos. Mesmo que os processos atuais sejam informais, é preciso saber quais os pontos fortes e fracos deles. Portanto, sem o entendimento destes, é difícil estabelecer prioridades de melhoria. Contudo, é preciso diagnosticar, para poder receitar. Além disto, como já foi citado, muitas vezes existem, dentro da própria organização, pessoas com boas ideias para resolver os problemas atuais.

Nota-se (PÁDUA, 2012) melhorias de processos requerem investimentos significativos. É preciso ter pelo menos um pequeno grupo de pessoas cujo foco seja a melhoria dos processos, e não problemas específicos de cada projeto. Contudo, treinamento é fundamental, a melhoria dos processos envolve a absorção de conceitos que não são triviais, sobretudo e que vão até contra a intuição de algumas pessoas. Entretanto, o programa de treinamento tem de ter planejamento, recursos, cronograma e obrigatoriedade. Algumas ferramentas têm de ser adquiridas e implantadas.

Para (PÁDUA, 2012) o amadurecimento dos processos se faz passo a passo. Para ser viável, a melhoria deve ser feita em etapas planejadas. Entretanto, o processo de melhoria dos processos de software deve também ter pontos de controle onde a alta direção da organização possa decidir sobre seus rumos. Contudo, estes pontos podem corresponder a estágios intermediários de capacitação. Porém, a experiência da indústria de software deu origem a vários modelos de capacitação, que propõem áreas prioritárias para investimento em melhoria de processos.

Contudo, são realmente importantes os conceitos de mudança eficaz, melhorias de processos e amadurecimento de processo? Para a estratégia e organização dos processos das rotinas?

2.10 A IMPORTANCIA DE CONHENCER OS PROCESSOS

Segundo (PRESSMAN, 2011) indivíduos, negócios e governos dependem, de forma crescente, de software para decisões estratégicas e táticas, assim como para controle e para operações cotidianas. Contudo, se o software falhar, as pessoas e as principais empresas poderão vivenciar desde pequenos inconvenientes a falhas catastróficas.

Segundo (FERREIRA, 2016), “o relacionamento das pessoas com o seu trabalho estabelece as bases comportamentais necessárias para a mudança organizacional, e depende mais das atitudes e das decisões individuais do que da tecnologia e das decisões superiores”.

Para (PRÉVE; MORITZ; PEREIRA, 2010) os processos de tomadas de decisão são constantes, entretanto no dia a dia organizacional e a todo o momento as pessoas estão sendo colocadas, sobretudo em uma situação em que é necessário analisar, investigar, optar e agir frente às poucas ou às muitas opções que lhes são fornecidas para decidir.

Segundo (FERREIRA, 2016) um dos desafios da boa gestão na atualidade é, com certeza, o domínio do conhecimento de seus processos organizacionais, que tem grande importância, sobretudo como subsídio para diversas ações na organização, tais como o gerenciamento do desempenho, a tomada de decisão, o dimensionamento da força de trabalho, contudo a desburocratização, a manutenção das rotinas, a melhoria dos serviços e produtos, a flexibilização organizacional, entre outros. Portanto, o gerenciamento dos processos permite uma visão sistêmica da organização, tratando-a como um conjunto de processos inter-relacionados, com foco nas expectativas ou requisitos dos clientes, usuários, cidadãos.

Para (FERREIRA, 2016), “para que decisões sejam tomadas. De nada adiantaria informações atrasadas e desatualizadas, embora corretas, ou informações atuais e corretas, mas para a pessoa errada”.

Segundo (SOMMERVILLE, 2011), existem quatro opções estratégicas:

A primeira opção se refere em, Descartar completamente o sistema. Essa opção deve ser escolhida quando o sistema não está mais contribuindo efetivamente para os processos dos negócios. Portanto, isso geralmente ocorre quando os

processos de negócios se alteram desde que o sistema foi instalado e já não são dependentes do sistema legado.

A segunda opção se refere em, Deixar o sistema inalterado e continuar com a manutenção regular. Entretanto, essa opção deve ser escolhida quando o sistema ainda é necessário, mas é bastante estável e os usuários do sistema fazem poucas solicitações de mudança.

A terceira opção se refere em, reestruturar o sistema para melhorar sua manutenibilidade. Entretanto, essa opção deve ser escolhida quando a qualidade do sistema foi degradada pelas mudanças, e novas mudanças para o novo sistema ainda estão sendo propostas. Portanto, esse processo pode incluir o desenvolvimento de novos componentes de interface, para que o sistema original possa trabalhar com outros sistemas mais novos.

A quanta opção se refere em, substituir a totalidade ou parte do sistema por um novo sistema. Entretanto, essa opção deve ser escolhida quando fatores como hardwares novos significam que o sistema antigo não pode continuar em operação ou, sobretudo quando sistemas de prateleira podem permitir o desenvolvimento do novo sistema a um custo razoável. Portanto, em muitos casos, uma estratégia de substituição evolutiva pode ser adotada, na qual, contudo sempre que possível, os componentes principais do sistema são substituídos por sistemas de prateleira com outros componentes reusados.

2.11 ORGANIZAÇÃO E CENTRALIZAÇÃO DOS PROCESSOS

Sobre o modelo de processos, (PRESSMAN, 2011), explica que: os modelos de processo prescritivos são aplicados há anos, num esforço para organizar e estruturar o desenvolvimento de software. Entretanto, cada um desses modelos sugere um fluxo de processos ligeiramente diferente, mas todos realizam o mesmo conjunto de atividades metodológicas genéricas: sobretudo comunicação, planejamento, modelagem, construção e emprego.

Para (MARKS, 2008), “a centralização, nas organizações, tem a ver com a distribuição”. Portanto, seja a empresa de porte pequeno, seja de porte médio, seja de porte grande, para funcionar bem ela necessita de uma estrutura organizacional adequada. Portanto, ou seja, as atividades importantes que nela devem ser desenvolvidas precisam estar previstas e devidamente organizadas, ou há

deficiência no funcionamento da organização.

Analisaremos agora a seguinte observação sobre departamentalizar de (MARKS, 2008), que diz o seguinte:

Segundo (MARKS, 2008) departamentalizar é dividir a empresa em órgãos ou unidades organizacionais. Entretanto, em cada órgão são realizadas atividades afins. Sobretudo por exemplo, no departamento financeiro são realizadas todas as atividades relacionadas com as finanças da empresa, contudo assim como no departamento de vendas são tratadas as questões relacionadas às vendas. Portanto, geralmente o nome do órgão corresponde ao que nele é feito. Um órgão pode ser subdividido em órgãos menores se isso for conveniente.

Será que poderíamos subdividir o código fonte do software para cada tipo de processo? Segundo (MARKS, 2008), “departamentalizar é uma maneira de organizar o processo” que se faz em forma “de trabalho na empresa. Na verdade, tudo o que se faz numa empresa é feito em forma de processo”. Contudo, processo tem “uma atividade inicial, outras intermediárias e uma atividade final”.

Segundo (MARKS, 2008) as atividades intermediárias geralmente realizam o processamento, ou seja, algum tipo de trabalho de transformação ou modificação. Entretanto, todo o processo tem início com pelo menos uma atividade, ou mais de uma, e termina, igualmente, com pelo menos uma ou mais de uma atividade. Portanto, nas atividades iniciais ocorrem as entradas do processo e nas finais ocorrem as saídas. Contudo, as entradas correspondem a tudo o que o processo recebe e as saídas, ao que ele fornece. Porém, o grande processo geral de uma empresa pode compor-se de um conjunto de processos menores.

Para (ARMELIN; SILVA; COLUCCI, 2016) um sistema empresarial constitui uma estrutura centralizada para uma organização e sobretudo garante que as informações possam ser compartilhadas por todas as funções da empresa e por todos os níveis de gerência para apoiar o gerenciamento de um negócio. Entretanto, os sistemas empresariais empregam um banco de dados operacional e de planejamento fundamentais que podem ser compartilhados por todos. Portanto, isso elimina os problemas de informações inconsistentes causados por múltiplos sistemas de processamento de transação, contudo que atuam somente em uma função do negócio ou em função de um departamento de uma organização.

Entretanto sobre sistema empresarial, segundo (ARMELIN; SILVA; COLUCCI, 2016) quanto aos seus benefícios podemos elencar:

- O benefício de auxílio na tomada de decisão empresarial.
- O benefício de aumento do valor agregado aos produtos, bens e serviços prestados pela empresa.
- O benefício de criação ou desenvolvimento de uma vantagem competitiva.
- O benefício de desenvolvimento e/ou fabricação de produtos com maior qualidade.
- O benefício de maior oportunidade de negócios, uma vez que se conhece o negócio da empresa.
- O benefício de maior rentabilidade.
- O benefício de maior precisão de informações, com isso, redução de erros.
- O benefício de diminuição de desperdícios e com isso, menores custos.

Para (SOMMERVILLE, 2011) existem fundamentos de engenharia de software que se aplicam a todos os tipos de sistemas de software:

1. Primeiramente eles devem ser desenvolvidos em um processo gerenciado e compreendido. Entretanto, a organização que desenvolve o software deve planejar o processo de desenvolvimento e ter idéias claras do que será produzido e quando estará finalizado. Portanto, é claro que processos diferentes são usados para tipos de software diferentes.

2. Segundo, confiança e desempenho são importantes para todos os tipos de sistema. Entretanto, o software deve se comportar conforme o esperado, sem falhas, e deve estar disponível para uso quando requerido. Portanto, deve ser seguro em sua operação e deve ser, tanto quanto possível, protegido contra-ataques externos. O sistema deve executar de forma eficiente e não deve desperdiçar recursos.

3. Terceiro, é importante entender e gerenciar a especificação e os requisitos de software (o que o software deve fazer). Entretanto, deve ser saber o que clientes e usuários esperam dele e deve gerenciar suas expectativas para que um sistema útil possa ser entregue dentro do orçamento e do cronograma.

4. Quarto deve ser fazer o melhor uso possível dos recursos existentes. Isso significa que, quando apropriado, deve se reusar o software já desenvolvido, em vez de escrever um novo.

Segundo (SOMMERVILLE, 2011) os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Entretanto, não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software. Portanto, os processos têm evoluído de maneira a tirarem melhor proveito das capacidades das pessoas em uma organização, bem como das características específicas do sistema em desenvolvimento. Contudo, para alguns sistemas, como sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado; para sistemas de negócios, sobretudo com requisitos que se alteram rapidamente, provavelmente será mais eficaz um processo menos formal e mais flexível.

Os processos de software, segundo (SOMMERVILLE, 2011) às vezes, são categorizados como dirigidos a planos ou processos ágeis. Entretanto, processos dirigidos a planos são aqueles em que todas as atividades são planejadas com antecedência, sobretudo o progresso é avaliado por comparação com o planejamento inicial. Porém, em processos ágeis, o planejamento é gradativo, e é mais fácil alterar o processo de maneira a refletir as necessidades de mudança dos clientes.

Para (SOMMERVILLE, 2011) embora não exista um processo ‘ideal’ de software, há espaço, em muitas organizações, para melhorias no processo de software. Entretanto, os processos podem incluir técnicas ultrapassadas ou não aproveitar as melhores práticas de engenharia de software da indústria. Contudo, de fato, muitas empresas ainda não se aproveitam dos métodos da engenharia de software em seu desenvolvimento de software.

Segundo (SOMMERVILLE, 2011) em organizações nas quais a diversidade de processos de software é reduzida, os processos de software podem ser melhorados pela padronização. Entretanto, isso possibilita uma melhor comunicação, além de redução no período de treinamento, e torna mais econômico o apoio ao processo automatizado. Portanto, a padronização também é um importante primeiro passo na introdução de novos métodos e técnicas de engenharia de software, assim como as boas práticas de engenharia de software.

Para uma melhor compreensão sobre organização de processos (PÁDUA, 2012) cita os 5 níveis de organização, que são:

- A organização nível 1 representa o estágio inicial dos produtores de software. Entretanto, ela utiliza processos informais e métodos ad hoc, às vezes descritos como caóticos. Portanto, muitas destas organizações são bem-sucedidas, já que o mercado de software é ainda extremamente tolerante em relação à má qualidade dos produtos. Porém, muitas vezes a qualidade do marketing pode ocultar deficiências técnicas, e existe pouca competição, em muitos setores deste mercado. A cultura no Nível Inicial é muito baseada no valor dos indivíduos. Entretanto, é comum a dependência em relação a heróis técnicos e gerenciais. Contudo, a organização nível 1 geralmente não é capaz de fazer estimativas de custo ou planos de projeto; se faz, não é capaz de cumpri-los. No entanto, as ferramentas não são integradas com os processos, e não são aplicadas com uniformidade pelos projetos. Geralmente, a codificação é a única fase dos processos de desenvolvimento que merece atenção. Contudo, engenharia de requisitos e desenho é fraco ou inexistente; mudanças de requisitos e de outros artefatos ocorrem sem controle. Porém, a instalação e manutenção costumam ser deficientes, sendo encaradas como atividades de pouca importância. Sobretudo, os gerentes destas organizações geralmente não entendem os verdadeiros problemas, por falta de processos que lhes dêem visibilidade real em relação ao progresso dos projetos. Entretanto, são comuns os casos de gerentes com formação exclusivamente administrativa, que não entendem os problemas técnicos dos projetos. Todavia, existem também aqueles que chegaram a gerentes como promoção da carreira técnica, e não têm a mínima formação em práticas gerenciais. Contudo, podem existir processos definidos no papel, que não são aplicados na realidade, ou são sempre contornados, com a cumplicidade e até a pressão dos gerentes. Porém, na melhor das hipóteses, os processos são seguidos quando os projetos estão em fase tranqüila; em crise, abandonam-se os métodos, e reverte-se à codificação desenfreada.

- A tônica da organização nível 2 é ser capaz de cumprir compromissos. Entretanto, no nível repetível, uma organização é capaz de assumir compromissos referentes a requisitos, prazos e custos com alta probabilidade de ser capaz de cumpri-los. Contudo, para (PÁDUA, 2012) requer o domínio das seguintes áreas chaves:

• Áreas como a gestão de requisitos permite definição e controle dos requisitos em que se baseiam os compromissos;

• Áreas como o planejamento de projetos prevê prazos e custos para cumprimento dos compromissos, como bases técnicas e não apenas intuitivas;

• Áreas como a supervisão e acompanhamento de projetos confere o atendimento dos compromissos, sobretudo comparando o conseguido com o planejamento, e acionando providências corretivas sempre que haja desvios significativos em relação aos compromissos;

• Áreas como a gestão da subcontratação cobra de organizações subcontratadas para desenvolver partes do software, contudo os mesmos padrões de qualidade que a organização principal oferece a seus clientes;

• Áreas do grupo de garantia da qualidade, que confere o cumprimento dos compromissos, de forma independente em relação aos projetos;

• Áreas como a gestão de configurações garante a consistência permanente dos resultados dos projetos, sobretudo entre si e com os requisitos, ao longo do projeto, mesmo quando ocorram alterações nos compromissos.

Segundo (PÁDUA, 2012) a organização nível 2 é disciplinada a nível dos projetos. Por isto, ela sabe estimar e controlar projetos semelhantes a projetos anteriores bem-sucedidos. Entretanto, ela corre riscos diante de vários tipos de mudanças a que as organizações estão sujeitas, tais como:

- Sujeitas as mudanças de ferramentas e métodos, trazidos pela evolução de tecnologia;

- Sujeitas as mudanças de tipos de produto, causadas por variações dos mercados;

- Sujeitas as mudanças de estrutura organizacional, causadas por diversos fatores da dinâmica das organizações.

Para (PÁDUA, 2012) o nível 3 conduz da gestão de projetos à engenharia de produtos. Entretanto, este nível de organização não repete simplesmente os sucessos de projetos anteriores, mas estabelece uma infra-instrutora de processos que permite a adaptação a vários tipos de mudanças. Contudo, este nível de organização requer o domínio das seguintes áreas chaves:

• domínio em estabelecimento formal de um grupo de processos de engenharia de software, responsável pelas atividades de desenvolvimento, melhoria e manutenção de processos de software;

• domínio em estabelecimento de um processo padrão de software em nível da organização, a partir do qual devem ser derivados os processos definidos para os projetos;

• domínio em estabelecimento de um programa de treinamento em processos de software, em nível da organização;

• domínio em gestão integrada dos projetos, baseada nos processos definidos para os projetos, com o uso de procedimentos documentados para gestão de tamanho, esforços, prazos e riscos;

• domínio em padronização em nível da organização dos métodos de engenharia de produtos de software, abrangendo engenharia de requisitos, testes, desenho, codificação e documentação de uso;

• domínio em coordenação entre os grupos que participam de projetos de sistemas, em nível da organização;

• domínio em coordenação de revisões técnicas a nível da organização.

Segundo (PÁDUA, 2012) a organização nível 3 sabe manter-se dentro do processo, mesmo durante as crises. Entretanto, as ferramentas passam a ser aplicadas de forma sistemática, padronizada e coerente com os processos. Contudo, com isto, passam a contribuir significativamente para melhoria da produtividade e qualidade. Por outro lado, o conhecimento dos processos, por parte da organização nível 3, ainda é basicamente qualitativo. Porém, existe uma base de dados de processos, povoada com os dados recolhidos dos projetos; sobretudo esta base é usada para gestão dos projetos, mas não é ainda aplicada, de forma sistemática e em nível da organização, para atingir metas quantitativas de desempenho de processo e de qualidade de produto.

Para (PÁDUA, 2012) a organização nível 4, o domínio dos processos de software evolui para uma forma quantitativa. Entretanto, isto não quer dizer que apenas organizações deste nível devam coletar métricas de processo. Contudo, todas as áreas chaves do CMM contêm pelo menos uma prática de medição e análise, que sugere métricas adequadas para medir o sucesso da implantação da respectiva área. Porém, a organização nível 3 constrói e mantém uma base de dados de processos. Entretanto, coleta de dados é uma atividade cara: é necessário definir com precisão e antecipação os dados que vão ser coletados. Contudo, estes dados têm de ser criticados, consistidos e normalizados para terem alta qualidade. Porém, a organização nível 4 é proficiente em coletar métricas e gerir a base de dados de processo, que é povoada e analisada por profissionais treinados. Sobretudo, além disto, sabe intervir nos processos para atingir metas de qualidade dos produtos. Contudo para (PÁDUA, 2012) este nível tem apenas duas áreas chaves:

• área de gestão quantitativa dos processos controla o desempenho dos processos usados pelos projetos;

• áreas de gestão da qualidade de software promovem o entendimento quantitativo da qualidade dos produtos de software, permitindo atingir metas quantitativas desejadas. Entretanto, a organização nível 4 passa da engenharia de produtos à qualidade de processos e produtos. Contudo, ela tem elementos para decidir, por exemplo, qual deve ser a fração de recursos dos projetos destinada à garantia da qualidade, sobretudo considerando o nível máximo de defeitos que se quer admitir nos produtos. Porém, este domínio quantitativo dos processos é necessário para atingir um estado de melhoria contínua.

Segundo (PÁDUA, 2012) a organização nível 5 atinge um estado em que os processos estão em melhoria contínua, sendo otimizados para as necessidades de cada momento. As seguintes áreas chaves são executadas:

• executar a prevenção dos defeitos, através da identificação e remoção das causas deles;

• executar a gestão da evolução tecnológica, com procedimentos sistemáticos de identificação, análise e introdução de tecnologia apropriada;

• executar o uso dos dados de processo para gestão das mudanças de processos, colocando-os em melhoria contínua.

Para (SOMMERVILLE, 2011) uma visão generalizada de que a melhor maneira para conseguir o melhor software era por meio de um planejamento cuidadoso do projeto, qualidade da segurança formalizada, sobretudo, do uso de métodos de análise e projeto apoiado por ferramentas CASE (Computer-Aided Software Engineering) e do processo de desenvolvimento de software rigoroso e controlado.

A respeito do gerenciamento do software a nível organizacional, para (SOMMERVILLE, 2011) o gerenciamento de qualidade está preocupado com o estabelecimento de um framework de processos organizacionais e padrões que levem a softwares de alta qualidade. Entretanto, isso significa que a equipe de gerenciamento de qualidade deve assumir a responsabilidade de definir os processos de desenvolvimento do software, sobretudo que serão usados e os padrões que devem ser usados no software, bem como a documentação relacionada”. Inclui também “os requisitos de sistema, projeto e código.

Falando de processo de desenvolvimento do sistema no quesito adaptação e reuso, para (SOMMERVILLE, 2011) o reuso de software é mais eficaz quando está previsto como parte de um programa de reuso de toda a organização. Entretanto, um programa de reuso envolve a criação de ativos reusáveis e a adaptação de processos de desenvolvimento para incorporar esses ativos no novo software.

Segundo (ARMELIN; SILVA; COLUCCI, 2016) “o uso da tecnologia da informação proporciona uma visão muito mais ampla sobre os processos da empresa e isso facilita, e muito, a eficácia e a eficiência”.

Contudo (PÁDUA, 2012), ressalta a maturidade das organizações e seus sintomas: A produção industrial de software é quase sempre uma atividade coletiva. Entretanto, alguns produtos são construídos inicialmente por indivíduos ou pequenas equipes. Na medida em que se tornam sucesso de mercado, passam a evoluir. Porém, a partir daí um número cada vez maior de pessoas passa a cuidar da manutenção e evolução dele. Contudo, por isso, quase todas as atividades de engenharia de software são empreendidas por organizações.

Para (PÁDUA, 2012) a maturidade de uma organização em engenharia de software mede o grau de competência, técnica e gerencial, que esta organização possui para produzir software de boa qualidade, dentro de prazos e custos razoáveis e previsíveis.

Ressaltando, segundo (PÁDUA, 2012) infelizmente para os profissionais, muitas organizações que produzem software são imaturas. Entretanto, isto ocorre tanto com organizações que produzem software, sobretudo como atividade fim, como com organizações para as quais o software é meio de apoio aos processos de negócio.

Segundo (PÁDUA, 2012) alguns sintomas identificam claramente as organizações imaturas, como:

• As organizações cujos projetos não são definidos com clareza. Atividades de desenvolvimento de software são disfarçadas de manutenção, ou mesmo realizadas sem nenhum marco formal. Entretanto, às vezes não se sabe ao certo quem é o responsável por um projeto, ou mesmo se uma atividade faz parte de algum projeto. Contudo, os clientes e usuários não sabem exatamente a quem se dirigir. Os gerentes têm dúvidas sobre o quê cobrar de quem.

• As organizações cujas pessoas não recebem o treinamento necessário. Ou não existe disponibilidade de tempo para treinamento, ou as pessoas se inscrevem no treinamento que bem entendem. Entretanto, os treinamentos são avaliados apenas quanto à satisfação dos treinados, se tanto. Porém, não se avalia o treinamento quanto ao benefício trazido para os projetos, e não se comparam benefícios com custos. Contudo, as pessoas trabalham em ambientes inadequados. Não existem planos claros de recrutamento, sobretudo remuneração e avaliação do desempenho. Não existe boa comunicação entre as pessoas.

• As organizações cujas ferramentas não ajudam realmente a resolver os problemas. As pessoas não têm acesso a ferramentas compatíveis com o estado da arte, ou têm as ferramentas que bem entendem. Entretanto, os usuários das ferramentas não recebem treinamento e orientação em grau satisfatório. Contudo, ferramentas são escolhidas de forma política, sem considerar avaliações técnicas e necessidades de padronização.

• As organizações cujos procedimentos e padrões, quando existem, são definidos e seguidos de forma burocrática. Entretanto, muitas vezes, o processo oficial, que existe nos documentos escritos, é rígido demais. Contudo, por outro lado, o processo real praticado é, com freqüência, muito diferente do processo oficial, e muito mais relaxado que este. Portanto, os gerentes são os primeiros a não levar os processos a sério.

3 **PROCEDIMENTOS METODOLÓGICOS**

De caráter formal exploratório, este trabalho, que discorre sobre o presente tema, como a engenharia de software pode melhorar o desenvolvimento de rotinas de processos e projetos do software, nesse novo propósito organizacional, é um produto da pesquisa, realizado com base em bibliografias de autores renomados e experiência profissional, cujo objetivo foi analisar e conceituar a importância do desenvolvimento e melhorias de rotinas, utilizando métodos e padrões da engenharia de software para desperta opiniões sobre a importância de centralizar rotinas estrategicamente para garantir um bom desenvolvimento dos processos de rotina do software, sistemas legados de organizações. Optou-se aqui, utilizar bibliografias de autores e suas obras, com diversos conceitos e diversas opiniões, como forma de levantar possíveis opiniões que se discutidas com antecedência pudesse ter efeito na qualidade no desenvolvimento das rotinas do software da organização, no caso a devida cautela no desenvolvimento do software, para garantir o processo de melhoria das rotinas do legado, garantindo assim a qualidade.

3.1 TIPO DE PESQUISA

Trata-se de uma pesquisa qualitativa de caráter exploratório com o propósito que, procura explorar e fornecer informações para futuras pesquisas relacionadas, na qual visa maior proximidade com o tema proposto.

3.2 COLETA DE DADOS

A técnica utilizada para a coleta de dados foi a observação direta, por meio de um questionário. As perguntas foram formuladas com base na literatura dos autores cujo foram citados durante o referencial. Segundo (MARCONI; LAKATOS, 2017), “a observação direta extensiva realiza-se através do questionário, do formulário, de medidas de opinião e atitudes e de técnicas mercadológicas”.

3.3 ANÁLISE DE DADOS

A análise dos dados foi feita com base no questionário e também na análise de conteúdo. Para (MARCONI; LAKATOS, 2017, p. 201), “questionário é um instrumento de coleta de dados”, auxiliador para uma coleta de dados, “constituído por uma série ordenada de perguntas, que devem ser respondidas por escrito e sem

a presença do entrevistador”. Contudo, “em geral, o pesquisador envia o questionário ao informante, pelo correio ou por um portador; depois de preenchido, o pesquisado devolve-o do mesmo modo”. Seguido conforme descrito o questionário elaborado, a “análise de conteúdo permite a descrição sistemática, objetiva e quantitativa do conteúdo da comunicação”. Entretanto, essa “análise de conteúdo, extrair generalizações com o propósito de produzir categorias conceituais que possam vir a ser operacionalizadas em um estudo subseqüente”.

3.4 PROCEDIMENTOS

Para a análise de conteúdo, foram coletadas informações no Google acadêmico https://scholar.google.com.br, Google livros https://books.google.com.br. Em relação a observação, foi observado no convívio profissional a dependência que o software, sistema legado da organização necessita de profissionais capacitados para a sua manutenção, em se tratando de experiência profissional foi feito um estudo de caso, foi criado e configurado um ambiente de desenvolvimento na linguagem plsql em um computador pessoal, onde software gratuito foram instalados para testar recursos da linguagem, como criação de rotinas, obedecendo os padrões da engenharia de software abordado no decorrer do estudo.

4 **DISCUSSÃO DOS RESULTADOS**

Neste capítulo apresentação e discussão dos resultados, serão apresentados e discutidos os principais resultados desta pesquisa ou investigação. Assim, tendo presente, na qual faz parte, do referencial, a revisão bibliográfica e com base nos dados colhido, especificamente a observação. O questionário e analise de conteúdo, procurou-se analisar e refletir sobre os padrões estudando pela a engenharia de software no quesito melhoria de sistemas, na qual, busca soluções para aperfeiçoar o sistema legado da organização. Veremos então como foi utilizado as técnicas de coleta de dados e análise de dados.

4.1 RESULTADOS

Entre as pessoas que responderam ao questionário das 10 perguntas, 30 estava ou já tinha concluído sua formação na área, ressaltando que 1 das 30 pessoas optou por responder uma das 10 perguntas do questionário que não tinha de 2 a 5 anos de experiência na área de tecnologia. Isso indica que para um grupo de pessoas a capacitação do profissional em engenharia de software é uma necessidade para a empresa. E esse entendimento é bom, para criar e melhorar rotinas do sistema legado da organização, de forma organizada e com qualidade.

4.2 OPINIÃO

Discutindo e interpretando os achados encontrados no resultado, nota-se que pessoas, profissionais da área de tecnologia, enxergam com bons olhos os benefícios que a engenharia de software pode trazer não só para o profissional em adquirir o conhecimento, quanto para o software da organização seu sistema legado. Os benefícios passados são também futuro já mencionado por sommerville, segundo (SOMMERVILLE, 2016), “a engenharia de software é, portanto, uma tecnologia de importância crítica para o futuro da humanidade”. Ressalta ainda o nosso dever de educar dizendo quer, “devemos continuar a educar engenheiros de software e a desenvolver a disciplina para podermos criar sistemas de software mais complexos”. Então se isto foi dito em sua obra em 2016, e com nossos resultados apontados, por que não asseguramos e sustentamos a opinião em 2020 como fortalecimentos do legado da engenharia de software.

4.3 ANÁLISE DOS RESULTADOS

Para (MARCONI; LAKATOS, 2017, p. 191), “do ponto de vista científico, a observação oferece uma série de vantagens e limitações, como as outras técnicas de pesquisa”. Uma dessas vantagens seria permitir a evidencia de dados, sobretudo “permite a evidência de dados não constantes do roteiro de entrevistas ou de questionários”. Contudo, “permite a coleta de dados sobre um conjunto de atitudes comportamentais típicas”. Enfim, “exige menos do observador do que as outras técnicas”.

Nota-se ainda (MARCONI; LAKATOS, 2017, p. 192), “a técnica da observação não estruturada ou assistemática”, contudo, “também denominada espontânea, informal, ordinária, simples, livre, ocasional e acidental”, entretanto, “consiste em recolher e registrar os fatos da realidade sem que o pesquisador utilize meios técnicos especiais ou precise fazer perguntas diretas”. Porém, “é mais empregada em estudos exploratórios e não tem planejamento e controle previamente elaborados”.

Segundo (MARCONI; LAKATOS, 2017, p. 223), “independentemente da(s) técnica(s) escolhida(s), deve-se descrever tanto a característica quanto a forma de sua aplicação, indicando, inclusive, como se pensa codificar e tabular os dados obtidos”. Para (SILVA, 2015, p. 59), “após a coleta de dados, o pesquisador irá organizá-los para poder analisar e, para esse fim, existem algumas técnicas”, de análise de dados, entre elas a análise de conteúdo.

Os resultados da pesquisa bibliográfica são as informações colhida com a investigação que deram suporte para o desenvolvimento das perguntas do questionário, segundo (MARCONI; LAKATOS, 2017) “a pesquisa bibliográfica é um apanhado geral sobre os principais trabalhos já realizados”, sobretudo, “revestidos de importância, por serem capazes de fornecer dados atuais e relevantes relacionados com o tema”. Sem a leitura da literatura das obras citadas na bibliografia não seria possível ter embasamentos para elaborar as perguntas do questionário, “o estudo da literatura pertinente pode ajudar a planificação do trabalho”, contudo, “evitar publicações e certos erros, e representa uma fonte indispensável de informações, podendo até orientar as indagações.

4.4 COLETA DE DADOS

Segundo (MARCONI; LAKATOS, 2017), “para obtenção de dados podem ser utilizados três procedimentos: pesquisa documental, pesquisa bibliográfica e contatos diretos”. As referências bibliográficas apresentadas ao longo do referencial foram coletadas no Google acadêmico e Google livros, onde os achados se ressumem em obras de diversos autores. Para colocar em prática as idéias sobre os padrões de engenharia de software na aplicabilidade do desenvolvimento de rotinas para a boa qualidade do software, foi sendo estudando, observado e revisado ao logo de meses as idéias apresentadas nas obras de literatura dos autores, sendo comparadas diversas opiniões sobre o mesmo assunto, e colocadas explicitamente no trabalho.

A pesquisa bibliográfica é um apanhado geral sobre os principais trabalhos já realizados, revestidos de importância, por serem capazes de fornecer dados atuais e relevantes relacionados com o tema. O estudo da literatura pertinente pode ajudar a planificação do trabalho, evitar publicações e certos erros, e representa uma fonte indispensável de informações, podendo até orientar as indagações (MARCONI; LAKATOS, 2017, p. 158).

A escolha desses dados para a análise dessa pesquisa partiu do questionário elaborado no Google forms, onde foi enviado um link para responder ao questionário, para cada e-mail dos participantes, contendo dez perguntas básicas, feito com 30 pessoas, cada equipe contendo 10 pessoas, profissionais atuantes em desenvolvimento de rotinas de processos, projetos e melhorias constantes de sistemas legados e software organizacional. Funcionários da empresa ivia, nas quais eram ou estavam se formando na área de tecnologia.

Segundo o site da organização, a empresa, fundada em 1996, a IVIA é uma empresa especializada em tecnologia da informação que desenvolve soluções que facilitem e ampliem os negócios de seus clientes. É uma das poucas empresas globais que possui avaliação ISO 9001, MPS.BR e CMMI, tendo sido eleita, pelo décimo ano consecutivo, como uma das melhores empresas para trabalhar no Brasil, segundo o Great Place to Work ®. Como uma empresa contemporânea, a IVIA se preocupa com inovações tecnológicas e causas humanitárias.

Segundo o site da organização, a empresa, fundada em 1996, a IVIA é uma empresa especializada em tecnologia da informação que desenvolve soluções que

facilitem e ampliem os negócios de seus clientes. É uma das poucas empresas globais que possui avaliação ISO 9001, MPS.BR e CMMI, tendo sido eleita, pelo décimo ano consecutivo, como uma das melhores empresas para trabalhar no Brasil, segundo o Great Place to Work ®. Como uma empresa contemporânea, a IVIA se preocupa com inovações tecnológicas e causas humanitárias.

Essas perguntas embora pessoais, porém, dentro do perfil da área em questão e com base nos autores e suas obras, me elevaram a curiosidade e vontade de buscar respostas para a pesquisa e investigação. Motivaram e continuam me motivando. Pois, foi levantado pontos importante para solucionar o problema que é a qualidade do software empresarial. Essas respostas foram cruciais para se levantar a relevância do tema pesquisado e escolha da fonte de pesquisa e coleta de dados. Entretanto, Segundo (MARCONI; LAKATOS, 2017), a técnica de observação direta apresenta “medidas de opinião e de atitudes - instrumento de "padronização", por meio do qual se pode assegurar a equivalência de diferentes opiniões e atitudes, com a finalidade de compará-las”. Contudo, no tópico seguinte as respostas e analise do questionário citado, abaixo as perguntas mencionadas no questionário.

1) Contudo o modelos de processo de software, segundo (SOMMERVILLE, 2016, p. 19), “um modelo de processo de software é uma representação simplificada de um processo de software. Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele”. Certamente talvez fosse interessante concordar com essa citação sobre modelos de processos de software? Sim ou não?

2) Portanto (SOMMERVILLE, 2016, p. 19) “esses modelos genéricos não são descrições definitivas dos processos de software. Pelo contrário, são abstrações que podem ser usadas para explicar diferentes abordagens de desenvolvimento de software”. Sobretudo como profissional, seria, contudo interessante uma nova abordagem para auxiliá-lo no processo de melhoria de rotinas do software da organização? Sim ou não?

3) Segundo (SOMMERVILLE, 2016, p. 19) faz uma abordagem a três modelos de processos, o “modelo em cascata (especificação de requisitos, projeto de software, implementação, teste), desenvolvimento incremental (O sistema é desenvolvido como uma série de versões (incrementos), de maneira que cada versão adiciona funcionalidade à anterior), engenharia de software orientada a reuso (O processo de desenvolvimento do sistema concentra-se na integração desses

componentes em um sistema já existente em vez de desenvolver um sistema a partir do zero)". Conclui-se que seria interessante implementar um conceito de organização de código fonte levando em conta todos esses modelos já existentes? Sim ou não?

4) É da área de engenharia de software ou de outras relacionadas ao desenvolvimento de software? Responda sim ou não.

5) Seria formando na área de engenharia de software ou de outras relacionadas ao desenvolvimento de software? Responda sim ou não.

6) Teria 2, 5 ou mais anos de experiência profissional na área de desenvolvimento de software ou áreas relacionadas? Responda sim ou não.

7) Seria maior de 18 anos? Responda sim ou não.

8) Saberia fazer de certa forma, levantamento de requisitos ou testes de software ou reuso de código fonte ou alguma etapas de processo de implementação de um software? Sim ou não?

9) Vivenciou sim ou não, profissionalmente algum desses modelos citados no questionário? Responda sim ou não.

10) De certa forma a vivência profissional e experiência no ramo de atividade em questão, um modelo de processo ou ciclo de vida de um software já apresentou falhas em seus processos e atividades de desenvolvimento, operação e manutenção? Sim ou não?

4.5 ANÁLISE DOS DADOS

Para (MARCONI; LAKATOS, 2017, p. 114), "uma vez manipulados os dados e obtidos os resultados, o passo seguinte é a análise e interpretação dos mesmos, constituindo-se ambas no núcleo central da pesquisa". Sobretudo, "é a tentativa de evidenciar as relações existentes entre o fenômeno estudado e outros fatores".

As respostas do questionário são sucintas, representadas por duas opções, sim ou não. Onde é representado pela a cor: vermelha para não e azul para sim. As análises dos dados coletados foram conforme as respostas, onde de trinta

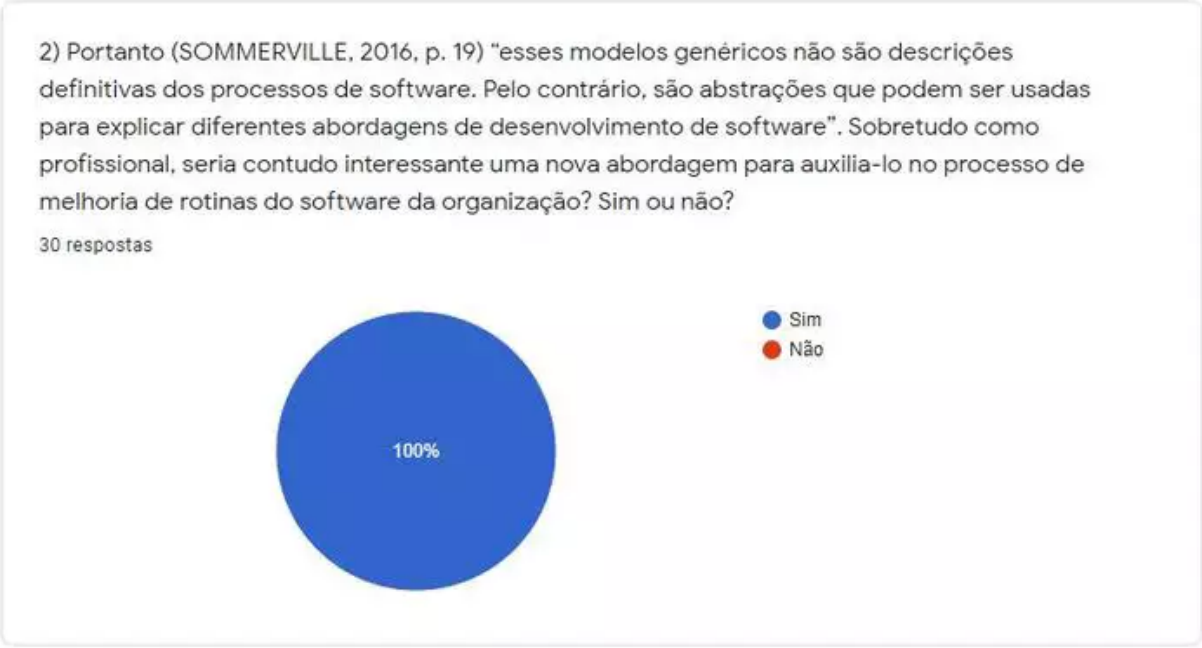
participantes, foram obtidos de dez perguntas 100% de aproveitamento. Onde a primeira pergunta teve 100% de resultado (sim) e 0% de (não), a segunda pergunta teve 100% de resultado (sim) e 0% de (não), a terceira pergunta teve 100% de resultado (sim), a quarta pergunta teve 100% de resultado (sim) e 0% de (não), a quinta pergunta teve 100% de resultado (sim) e 0% de (não), a sexta pergunta teve 83,3% de resultado (sim) e 16,7% de (não), a sétima pergunta teve 100% de resultado (sim) e 0% de (não), a oitava pergunta teve 100% de resultado (sim) e 0% de (não), a nona pergunta teve 100% de resultado (sim) e 0% de (não), a décima e última pergunta teve 100% de resultado (sim) e 0% de (não). Os dados foram analisados com a análise de conteúdo, segundo (CHIZZOTTI, 2016, p. 98), “Análise de conteúdo é um método de tratamento e análise de informações, colhidas por meio de técnicas de coleta de dados, consubstanciadas em um documento”. Essa técnica escolhida analise de conteúdo, “a técnica se aplica à análise de textos escritos ou de qualquer comunicação (oral, visual, gestual) reduzida a um texto ou documento”. Segundo (SILVA; FOSSÁ, 2015, p. 5) “destaca-se também, que a análise de conteúdo, enquanto conjunto de técnicas de análise de comunicações, ao longo dos anos”, houve alterações, onde “sofreu reformulações desde os primeiros preceitos até os dias atuais, com uma análise mais contemporânea, influenciada pelo uso do computador”. Porém, “hoje em dia, existem alguns softwares que auxiliam, principalmente, nos processos de organização do material e codificação dos dados”. Firmando-se nas palavras ditas, estaremos utilizando esses softwares de auxilio como o Google docs como fonte de coleta dos dados. Continuando, para (BARDIN, 2011), é “um conjunto de técnicas de análise de comunicação”, complementando, essa técnica “contem informação sobre o comportamento humano atestado por uma fonte documental”.

O seguinte endereço https://forms.gle/1kFTjFwJVQkT3skx8 encontra-se as perguntas do questionário.

Figura 3 — Certamente talvez seria interessante concordar com essa citação sobre modelos de processos de software? Sim ou não?

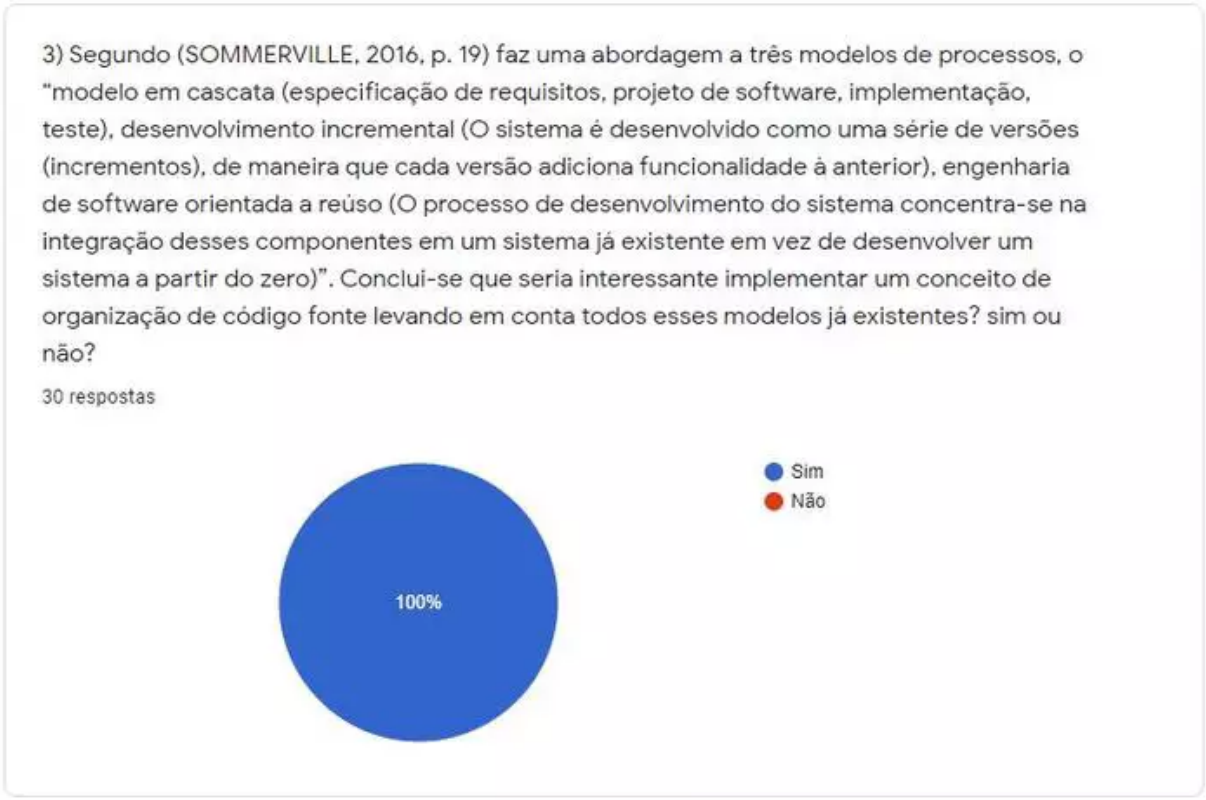
Fonte: O autor (2021)

Figura 4 — Sobretudo como profissional, seria, contudo interessante uma nova abordagem para auxiliá-lo no processo de melhoria de rotinas do software da organização? Sim ou não?



Fonte: O autor (2021)

Figura 5 — Conclui-se que seria interessante implementar um conceito de organização de código fonte levando em conta todos esses modelos já existentes? Sim ou não?



Fonte: O autor (2021)

Figura 6 — É da área de engenharia de software ou de outras relacionadas ao desenvolvimento de software? Responda sim ou não.

Fonte: O autor (2021)

Figura 7 — Seria formando na área de engenharia de software ou de outras relacionadas ao desenvolvimento de software? Responda sim ou não.

Fonte: O autor (2021)

Figura 8 — Teria 2, 5 ou mais anos de experiência profissional na área de desenvolvimento de software ou áreas relacionadas? Responda sim ou não.

Fonte: O autor (2021)

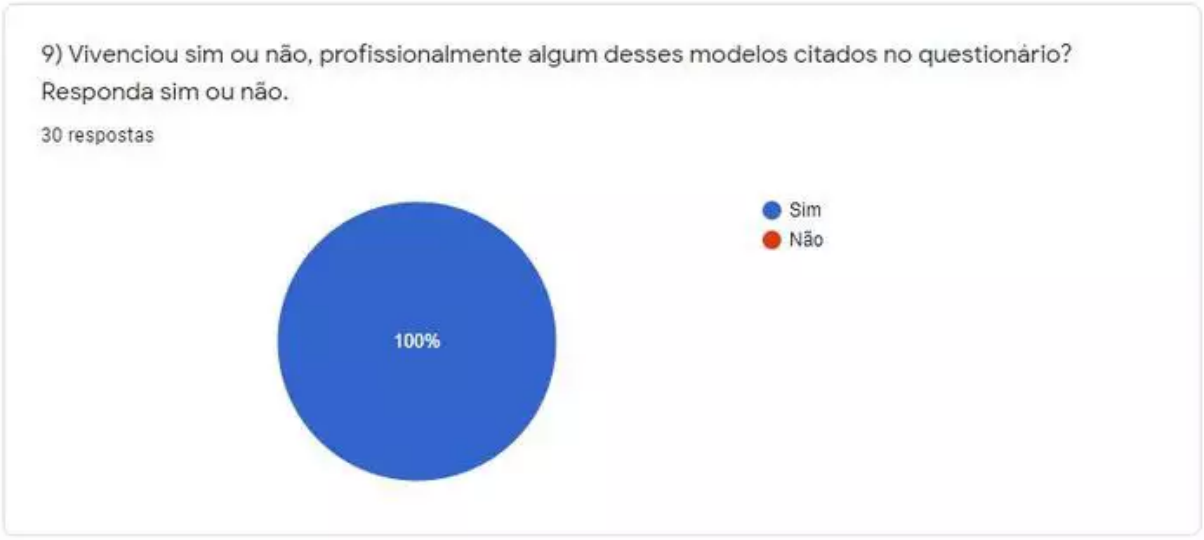
Figura 9 — Seria maior de 18 anos? Responda sim ou não.

Fonte: O autor (2021)

Figura 10 — Saberia fazer de certa forma, levantamento de requisitos ou testes de software ou reuso de código fonte ou alguma etapas de processo de implementação de um software? Sim ou não?

Fonte: O autor (2021)

Figura 11 — Vivenciou sim ou não, profissionalmente algum desses modelos citados no questionário? Responda sim ou não.



Fonte: O autor (2021)

Figura 12 — De certa forma a vivência profissional e experiência no ramo de atividade em questão, um modelo de processo ou ciclo de vida de um software já apresentou falhas em seus processos e atividades de desenvolvimento, operação e manutenção? Sim ou não?

Fonte: O autor (2021)

Tabela 1 — Quantidade de respostas por opção

Opções	Quantidade
Sim	295
Não	5
Em Branco	0
Total	300

Fonte: O autor (2021)

Quadro 1 — Cores de respostas por opções de seleção

Cor	Opção
Azul	Sim
Vermelho	Não

Fonte: O autor (2021)

Tabela 2 — Porcentagem por opções de respostas selecionadas

Questão	Resposta Sim	Resposta Não
1	100 %	0 %
2	100 %	0 %
3	100 %	0 %
4	100 %	0 %
5	100 %	0 %
6	83,3 %	16,7 %
7	100 %	0 %
8	100 %	0 %
9	100 %	0 %
10	100 %	0 %

Fonte: O autor (2021)

4.6 ESTUDO DE CASO

O estudo de caso aprofunda o que foi mencionado no tópico procedimento, às ferramentas utilizadas para viabilizar a criação de um novo propósito organizacional

onde organizar o código fonte da aplicação e descobrir possíveis falhas de modelos de processos de uma rotina de um processo de desenvolvimento foi o foco do estudo, com base na linguagem plsql, onde o ambiente de desenvolvimento foi construído em cima de um programa de computador de nome virtual Box da empresa Oracle com sua versão gratuita. Instalado em um computador pessoal, foi virtualizado dentro desse software um sistema operacional de nome Windows XP, é dentro do sistema virtualizado, foram instalados dois softwares em suas versões gratuitas para desenvolvedor também da empresa Oracle, software esses de nome Oracle forms, responsável para o desenvolvimento das telas para interação do usuário final e banco de dados Oracle, responsável em guardar as informações e organizar e centralizar o código fonte da aplicação. Onde é possível trabalhar com uma linguagem especifica desse software de desenvolvimento, a plsql. Toda essa estrutura é denominada são chamadas de máquina virtual.

Os profissionais participantes da pesquisa do questionário atuam na analise, desenvolvimento e testes do sistema legado, cada equipe contem 10 pessoas ou mais, os setores são diversos. Mais simplificando temos o cadastro de beneficiários e a movimentação desses dados cadastrais dos beneficiários.

Seria possível refletir na seguinte idéia de modelo de processo? O modelo cascata é a base. Poderemos representar o modelo cascata seguido de especificação → projeto → implementação → validação → manutenção. Lembre-se o modelo pode ser ligado e desligado na etapa de desenvolvimento para serem reanalisadas como ponto de parada, refazendo todas essas perguntas a seguir.

Segundo (SOMMERVILLE, 2016, p. 168), “existem sete atividades de desenvolvimento fundamentais, engenharia de requisitos, projeto arquitetônico, particionamento de requisitos, engenharia do subsistema, integração do sistema, teste do sistema, implantação do sistema”. Partindo desse principio:

Pressuponha-se assim:

1. Requirements engineering → Como posso processeguir?
2. Architectural design → Seria assim, dessa maneira?
3. Requirements partitioning → Precisaria interagir?
4. Subsystem engineering → Quais seriam as regras?
5. System integration → Poderiam juntar integrar as rotinas?
6. System testing → Seria viável apresentar?
7. System deployment → Poderia disponibilizar?

Preserva-se sempre o nome da etapa utilizada dentro dos padrões da engenharia de software para não ficar confuso, não é viável sair dos termos padrões já existentes na engenharia, nesse caso seguido de um termo auxiliar. Para (SOMMERVILLE, 2016, p. 168), “um dos aspectos mais confusos da engenharia de sistemas é que as empresas usam terminologia diferente para cada etapa do processo”.

4.6.1 **Alguns pontos importantes**

Como seria, o que teria um cadastro de beneficiário? Segundo (SOMMERVILLE, 2016) “engenharia de requisitos é o processo de refinar, analisar e documentar os requisitos de alto nível e de negócios identificados no projeto conceitual”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Assim, seguindo essa representação? Para (SOMMERVILLE, 2016) “o projeto arquitetônico se sobrepõe significativamente à engenharia de requisitos processo”. Entretanto, “o processo envolve o estabelecimento da arquitetura geral do sistema, identificar os diferentes componentes do sistema e compreender a relação navios entre eles”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Digitando, realizando o cadastro dessa forma? Segundo (SOMMERVILLE, 2016) “o particionamento de requisitos está preocupado em decidir quais subsistemas (identificados na arquitetura do sistema) são responsáveis pela implementação do sistema requisitos”. Entretanto, “os requisitos podem ter que ser alocados para hardware, software ou processos operacionais e priorizados para implementação”. Portanto, “idealmente, você deveria alocar requisitos para subsistemas individuais de modo que a implementação de um requisito crítico não precisa de colaboração de subsistema”. Contudo, “no entanto, isso não é sempre possível”. Sobretudo, “nesta fase, você também decide sobre os processos operacionais e sobre como eles são usados na implementação de requisitos”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Quais regras sofreram mudanças e, ou quais novas regras? Segundo (SOMMERVILLE, 2016) “a engenharia do subsistema envolve o desenvolvimento

dos componentes de software do sistema tem, configurando hardware e software de prateleira, projetando, se necessário”, sobretudo, “hardware para fins especiais, definindo os processos operacionais para o sistema, e redesenhar processos de negócios essenciais”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Seria o desenvolvimento de tudo que foi levantado, união das informações? Para (SOMMERVILLE, 2016) “a integração do sistema é o processo de reunir elementos do sistema para criar um novo sistema”. Entretanto, “só então as propriedades do sistema emergente se tornam aparentes”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Fazem-se testes das funcionalidades? Segundo (SOMMERVILLE, 2016) “o teste do sistema é uma atividade estendida onde todo o sistema é testado e apresenta problemas são expostos. Entretanto, “as fases de engenharia do subsistema e integração do sistema são reentradas para reparar esses problemas, ajustar o desempenho do sistema e implementar novos requisitos”. Contudo, “o teste de sistema pode envolver testes pelo desenvolvedor do sistema e teste de aceitação do usuário pela organização que adquiriu o sistema”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Ponto elevado "go live", disponibilizar as novas funcionalidades para os usuários utilizarem? Segundo (SOMMERVILLE, 2016) “a implantação do sistema é o processo de disponibilizar o sistema para seus usuários, transferir dados de sistemas existentes e estabelecer comunicações com outros sistemas no ambiente”. Entretanto, “o processo culmina com um "go live", após quais usuários passam a usar o sistema para apoiar seu trabalho”. Acredita-se que nesse sentido representa-se a base para exemplificar nosso termo auxiliar.

Para (SOMMERVILLE, 2016) “embora o processo geral seja orientado por planos, os processos de desenvolvimento de requisitos, o projeto e o projeto do sistema estão inextricavelmente ligados”.

4.6.2 **Protótipo**

Sobre a linguagem estudada e pesquisada, segundo (GONÇALVES, 2015) “a sigla PL/SQL significa Procedural Language Structured Query Language, ou seja, trata-se de uma linguagem procedural tendo como base a SQL (Linguagem de

Consulta Estruturada)". No entanto, (PRICE, 2008), indica que "o PL/SQL, que é baseado na linguagem SQL e permite escrever programas armazenados no banco de dados contendo instruções SQL. O PL/SQL contém construções de programação padrão".

Para (FERNANDES, 2002) a PL/SQL como "uma linguagem procedural da Oracle que estende a SQL com comando que permitem a criação de procedimentos de programação". Entretanto, o conceito de PL/SQL abrange uma estrutura procedural onde visa com base no padrão SQL sua extensão de linguagem. Nota-se (FERNANDES, 2002) "a linguagem permite a declaração de constantes, variáveis, subprogramas (procedures e funções), que favorecem a estruturação de código, e possui mecanismos para controle de erros de execução". Continuando, as centralizações desses subprogramas podem representar uma proposta de centralização e organização dos processos do sistema legado da organização? Portanto, "incorpora os novos conceitos de objeto, encapsulamento e, ainda, permite a interface com rotinas escritas em outras linguagens".

Segundo (PRICE, 2008), "os programas em PL/SQL são divididos em estruturas conhecidas como blocos, com cada bloco contendo instruções PL/SQL e SQL", nota-se (GONÇALVES, 2015), "a linguagem PL/SQL trabalha em blocos de comando. Dentro de bloco podemos ter outros blocos, que neste caso são chamados de sub-blocos". Ressalta ainda que, "quando queremos escrever um programa para um determinado fim", contudo, "utilizamos blocos para estruturar os comandos e a forma de como este programa vai se comportar". Por isso, "um bloco PL/SQL é iniciado pela expressão begin e é finalizada por end. Estas duas expressões determinam a área do nosso bloco". Segundo (FERNANDES, 2002), "a PL/SQL é estruturada em blocos. Cada bloco pode conter outros blocos". Por isso, "em cada um desses blocos, podemos declarar variáveis que deixam de existir quando o bloco termina".

Para (FERNANDES, 2002) a estrutura de um bloco PL/SQL é composta de três partes: A primeira se refere-se a uma parte declarativa, onde definimos as variáveis locais àquele bloco. A segunda se refere a uma parte de lógica, onde definimos a ação que aquele bloco deve realizar, incluindo a declaração de outros blocos subordinados (ou embutidos) a este. A terceira se refere a uma parte de tratamento de erros, que permite que tenhamos acesso ao erro ocorrido e a determinação de uma ação de correção. Entretanto, nessa parte, também podemos declarar outros blocos subordinados.

Segundo (GONÇALVES, 2015) “além das expressões de delimitação do bloco, também podemos ter o declare que é utilizado para delimitar uma área para declaração de variáveis que serão utilizadas”, sobretudo “pela aplicação e também a expressão exception, que delimita uma área para tratamento de erros”. Contudo, “basicamente, um bloco PL/SQL é composto por: área de declaração de variáveis (declare), área de escopo para inserção de comandos e demais sub-blocos (begin-end), área de tratamento de erros”.

Para (GONÇALVES, 2015) “quando temos dentro da aplicação comandos SQL distintos, eles são enviados um a um para o servidor do banco de dados”. Entretanto, “dessa forma, a aplicação envia um comando para o servidor, espera a resposta e depois envia outro”. Portanto, “quando temos blocos PL/SQL, eles são enviados por completo ao servidor do banco de dados, não importando o seu teor”. Contudo, “dentro deste bloco podemos ter vários comandos SQL e demais estruturas em PL/SQL”. Sobretudo, “desse modo economizamos tempo, pois a aplicação envia de uma só vez todas as solicitações, e o número de respostas esperadas também reduzem muito”. Porém, “reduzindo o número de respostas e tráfego de informações entre aplicação e o servidor do banco de dados aumentamos a chance de ganho de desempenho, principalmente se esta comunicação depender de uma rede cliente x servidor”.

Segundo (PRICE, 2008) “por padrão, cada unidade de programa PL/SQL é compilada em código legível pela máquina, na forma intermediária”. Entretanto, “esse código legível pela máquina é armazenado no banco de dados e interpretado sempre que o código é executado”. Contudo, “com a compilação nativa PL/SQL, o PL/SQL é transformado em código nativo e armazenado em bibliotecas compartilhadas”. Sobretudo, “o código nativo é executado muito mais rapidamente do que o código intermediário, pois não precisa ser interpretado antes da execução”.

Segundo (GONÇALVES, 2015) “a linguagem PL/SQL trabalha em blocos de comando”. Entretanto, “dentro de bloco podemos ter outros blocos, que neste caso são chamados de sub-blocos”. Contudo, “quando queremos escrever um programa para um determinado fim, utilizamos blocos para estruturar os comandos e a forma de como este programa vai se comportar”.

Sobretudo a programação em blocos, nota-se (GONÇALVES, 2015) “torna os programas mais estruturados e limpos. principalmente, quando utilizamos vários

sub-blocos para a realização de determinadas tarefas distintas", entretanto, "como a seleção de dados, uma atualização ou exclusão". Porém, "fazendo desta forma, é possível tratar cada bloco buscando uma programação mais lógica e que possa fornecer informações sobre os comandos contidos nele ou até mesmo identificar possíveis erros que possam surgir".

Segundo (PRICE, 2008) "toda instrução é terminada por um ponto-e-vírgula (;) e um bloco PL/SQL é terminado com o caractere de barra normal (/)". Para (GONÇALVES, 2015), "através do comando barra / podemos executar um bloco PL/SQL".

Segundo (GONÇALVES, 2015), "para construir um programa em PL/SQL temos que trabalhar em nível de bloco". Entretanto, "assim sendo, a linguagem PL/SQL permite adicionar, dentro das estruturas destes blocos, todo e qualquer recurso para que este programa possa executar ações ou procedimentos servindo". Porém, "dentro dos blocos é possível declarar variáveis e constantes, executar comandos DML (select, delete, update e insert), executar procedimentos armazenados, funções, utilizar estruturas de repetição, estruturas de condição", sobretudo, "além do uso de operadores relacionais e numéricos". Nota-se (PRICE, 2008), "os tipos PL/SQL são semelhantes aos tipos de coluna de banco de dados".

Acredita-se que seja necessário:

1 - Desenvolver uma tabela para registro de rotinas.
2 - Desenvolver uma tabela para cadastro de dados.
3 - Desenvolver uma tela padrão para digitação dos registros de rotinas.
4 - Desenvolver uma tela padrão para digitação de cadastro de dados.
5 - Desenvolver novo campo na tabela cadastro de dados.
6 - Desenvolver novo campo na tela padrão para digitação de cadastro de dados.
7 - Desenvolver nova rotina programável de validação de dados cadastrais.
8 - Desenvolver nova rotina programável para adição e organização de regra.
9 - Desenvolver e centraliza e organizar as rotinas de adição e validação.
10 - Desenvolver código programável para chamar as rotinas de cadastro.
11 - Desenvolver regra franco brasil um.
12 - Desenvolver e visualizar teste de regra franco brasil um.
13 - Desenvolver uma rotina de trabalho franco brasil um.
14 - Desenvolver e melhorar a rotina de trabalho franco brasil um.
15 - Desenvolver novo cadastro organizado para a rotina franco brasil um.

16 - Desenvolver regra franco brasil dois.
17 - Desenvolver e continuar regra franco brasil dois.
18 - Desenvolver novo cadastro organizado para a rotina franco brasil dois.
19 - Desenvolver e visualizar teste de regra franco brasil dois.

Dessa maneira representaremos a rotina do software organizada e centralizada com uma simples seqüência de controle, esse conceito de desenvolvimento tornará a estrutura e rotina do sistema legado da organização mais organizada e de fácil entendimento e acesso, claro, fácil de compreender pelo os antigos e novos profissionais atuantes nos processos de melhorias. Acredita-se que o código é o centro da organização das rotinas, deveria ter uma estrutura para centralizar todas as rotinas de regras de negocio enquanto provavelmente fará necessário um código contendo as regras de cadastro centralizadas que são executadas quando ativas e quando, a rotina de registro é chamada executada e disparada pela aplicação.

Para exemplificar a aplicação do conceito do propósito foi elaborado um fluxo de processo relacional, modelo E-R (entidade, relacionamento e atributos) para fins de estudo e aplicabilidade da idéia da teoria do novo propósito organizacional, centralizador de rotinas o modelo da simples seqüência.

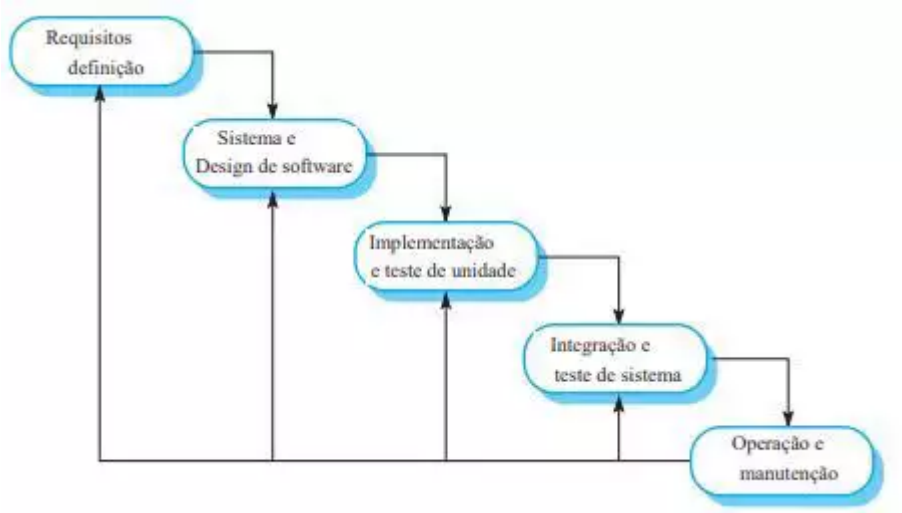
Segundo (HEUSER, 2009) o projeto de um banco de dados usualmente ocorre em três etapas. Em uma primeira analise, a primeira etapa surgi como uma modelagem conceitual, procura capturar formalmente os requisitos de informação de um banco de dados. Em uma segunda analise, a segunda etapa, contempla o projeto lógico, de forma objetiva definir, a nível de SGBD, as estruturas de dados que implementarão os requisitos identificados na modelagem conceitual. Em uma terceira analise, a terceira etapa, estende-se ao projeto físico, define parâmetros físicos de acesso ao BD, procurando otimizar a performance do sistema como um todo. Entretanto, resumindo, cobrem-se as duas primeiras etapas do ciclo de vida de um banco de dados, a da modelagem conceitual e a do projeto lógico.

Para (HEUSER, 2009) na modelagem conceitual, utiliza-se a abordagem entidade-relacionamento (ER) de Peter Chen, considerada hoje um padrão “de fato” de modelagem de dados. Entretanto, apresentam-se os conceitos e notações da abordagem ER, regras e heurísticas para construção de modelos. Referenciada ao projeto lógico, no qual cobre tanto o projeto propriamente dito (transformação de modelos ER em modelos relacionais), quanto à engenharia reversa de BD (extração

de modelo conceitual a partir de modelo lógico relacional ou de arquivos convencionais).

Pressuponha-se que o modelo em cascata de desenvolvimento de software foi utilizado como exemplo do decorrer do trabalho havia uma lacuna, ao adotar o modelo para o desenvolvimento das etapas dos processos das rotinas administrativas do grupo franco brasil, muitas vezes na etapa de integração e testes de sistemas um novo requisito que não foi previsto na etapa de requisitos e definição era pontuada, é muitas vezes o projeto não poderia seguir para a próxima etapa de operação e manutenção devido a esse problema. Havia a necessidade de um novo conceito para suprir essa necessidade, uma idéia para suprir e resolver essa falta foi desenvolvida, o propósito, idéia, modelo do PDSS (Propósito da Simples Seqüência).

Figura 13 — Representação genérica do modelo cascata



Fonte: Adaptado de SOMMERVILLE (2016)

Acredita-se que o propósito do PDSS, propósito da simples seqüência, organizador, centralizador de rotinas, irá auxiliar no processo de desenvolvimento e melhoria das rotinas do software, quando a problemática ocorre, acopla-se o PDSS, o analista responsável por esse processo, recebe o problema, organiza, testa, rever os requisitos e desenvolve e por mais uma vez rever os requisitos, testar e entrega o processo organizado, desacoplado o PDSS da etapa problemática e já organizada, pronto para seguir adiante. Acredita-se que o analista responsável deve ter os seguintes pontos: conhecer as rotinas do sistema, conhecer as regras de negócio do sistema, conhecer e entender da linguagem de programação do software.

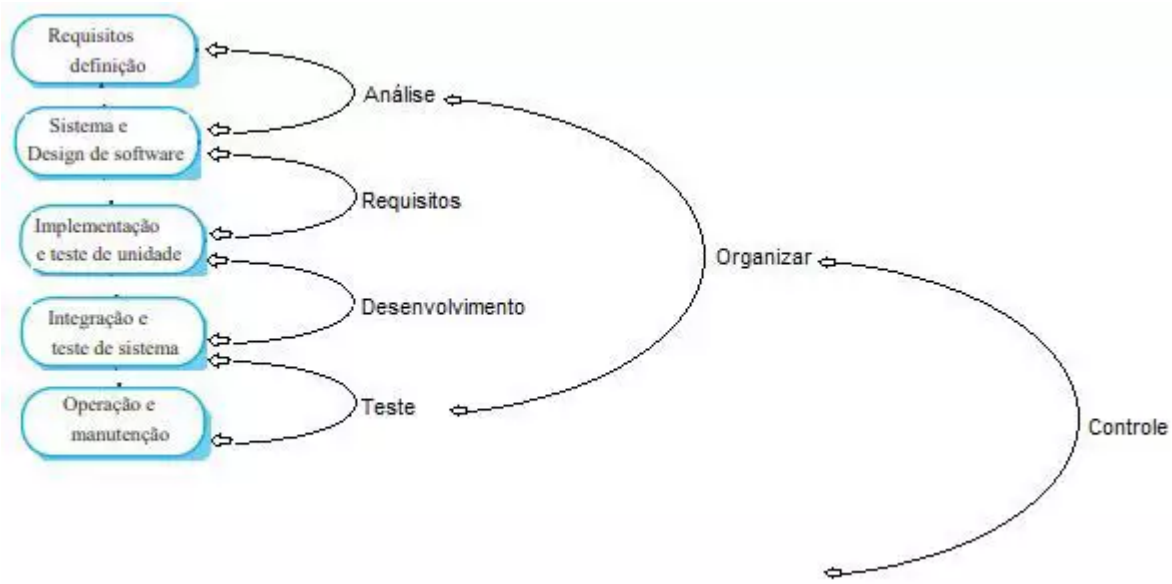
Figura 14 — Representação genérica do PDSS.

Fonte: O autor (2021)

Segundo (MARCONI; LAKATOS, 2017) para Karl R. Popper, o método científico hipotético-dedutivo parte de um problema (P1), contudo, ao qual se oferece uma espécie de solução provisória, uma teoria-tentativa (TT), passando-se depois a criticar a solução, com vista à eliminação do erro (EE). Entretanto, o método científico consiste na escolha de problemas interessantes e na crítica de nossas permanentes tentativas experimentais e provisórias de solucioná-los.

Contudo, o PDSS (Propósito da Simples Seqüência) foi proposto para auxiliar e tentar sanar os conflitos existentes na entrega das melhorias das rotinas de projetos do software, entretanto pretende-se com essa solução minimizar a contestação dos testes de processo, atraso das entregas e problemas de quebra de modelos de processos já existentes, para (MARCONI; LAKATOS, 2017) Popper defende esses momentos no processo investigatório, sobretudo, o problema surge, em geral, de conflitos ante expectativas e teorias existentes. Entretanto, a solução proposta consiste numa conjectura (nova teoria), portanto, dedução de conseqüências na forma de proposições passíveis de teste. Porém, os testes de falseamento são tentativas de refutação, entre outros meios, pela observação e experimentação.

Figura 15 — Representação genérica do modelo cascata e PDSS.



Fonte: Adaptado de Sommerville. (2016, p. 49)

Sendo assim proponho esse novo modelo organizacional e centralizador de rotinas a simples sequência com critérios e métodos abordados dentro dos padrões da engenharia de software.

5 **CONCLUSÃO**

Analisaremos nesse capitulo se o que foi proposto para o trabalho foi realmente contento, em forma, acontecimento e histórico.

Quando se iniciou o trabalho de pesquisa constatou-se que havia uma dúvida se realmente era importante organizar e centralizar as rotinas de um sistema legado, como a engenharia de software poderia melhorar o desenvolvimento de rotinas de processos e projetos do software com um novo propósito organizacional, se isso manteria seus processos com qualidade e organizados, mas como resolver isso se tinha uma escassez sobre o tema. Para resolver essa dúvida foi necessário realizar a pesquisa, procurar lacunas e tentar preenchê-las, reunindo as idéias de grandes escritores da literatura sobre os diversos temas e modelos da engenharia de software em um só trabalho. Então para preencher essas lacunas nasceu então o tema de pesquisa: simples sequencia para controlar e organizar o desenvolvimento de rotinas administrativas com engenharia de software: proposta de implementação no setor de registro hospitalar, junto com a seguinte pergunta, como a engenharia de software pode melhorar o desenvolvimento de rotinas de processos e projetos do software, nesse novo propósito organizacional? E por fim originou-se o trabalho aqui presente.

Diante disso a pesquisa teve como objetivo geral, analisar e propor um novo modelo que visa organizar e centralizar o desenvolvimento de rotinas do sistema hospitalar do estado do Ceará, com um possível e novo propósito organizacional. Com o objetivo de responder nos padrões da engenharia de software como realizar essa analise baseada nas idéias citadas dos diversos autores de obras e livros de engenharia de software. Constata-se que o objetivo geral foi atendido porque efetivamente o trabalho conseguiu responder a essa análise no tópico revisão da literatura em fundamentos de desenvolvimento e em discussão dos resultados foi apresentado um protótipo da visão organizacional das rotinas em desenvolvimento.

O objetivo especifico inicial era apresentar verificar a melhor sistemática de implantação para o propósito organizacional, ele foi atendido por apresentar no tópico revisão da literatura pontos importante sobre a implementação.

O segundo objetivo especifica era diagnosticar a estrutura da empresa hospitalar nos aspectos: organizacional, de pessoas e cada parte de trabalho e processos, por segurança de dados e informações foi apresentando um modelo

organizacional em resultados para exemplificar o conceito proposto em resultados, porém toda a revisão de literatura dentre as citações estavam sempre voltados para pontos estratégicos de melhoria ou vivenciados dentro do setor organizacional.

O terceiro objetivo especifica era implementar processos em cada setor estruturado para o grupo, foi descrito as principais características de desenvolvimento de software, ele foi atendido por apresentar no tópico revisão da literatura seus conceitos relacionados com o objetivo especifico.

O quarto objetivo especifica era averiguar a importância do desenvolvimento do software e suas principais características foram identificadas quais dificuldades existem no processo de desenvolvimento, ele foi atendido por apresentar no tópico revisão da literatura em qualidade de desenvolvimento.

O quinto e ultimo objetivo especifico era responder como organizar o desenvolvimento de rotinas do software ele foi atendidas por apresentar em topo o tópico revisão da literatura diversas citações sobre as possíveis respostas do objetivo.

E por fim, estes foram os objetivos que resultaram na elaboração do trabalho apresentado, mostrando possíveis soluções aproximadas do propósito.

A pesquisa partiu da hipótese de que os sistemas legados das organizações do Brasil passam por melhorias constantes, porém essas melhorias estão sendo feitas, às vezes, por profissionais que não conhecem os conceitos de refaturamento, organização de código fonte, falha em alguma etapa do modelo de processo escolhido ou por não ser adequar aos métodos estabelecidos nos desenvolvimentos adotados dentro do setor ou até mesmo abordando dentro da engenharia de software. Porque essa idéia de melhorar a rotina interna do software sem afetar a qualidade das existentes é essencial para manter a qualidade do sistema legado da organização, e fez com quer essa hipótese fosse levantada. Durante o trabalho verificou-se que pessoas da área de tecnologia estão querendo buscar conhecer uma melhor solução de desenvolvimento dentro da engenharia de software, essa busca pela a importância dos padrões de qualidade no processo de melhoria das rotinas do sistema legado organizacional é essencial para um melhor desenvolvimento da rotina, então se fez o teste da hipótese no capitulo procedimentos metodológicos onde a hipótese foi confirmada.

Em resposta ao problema em certa parte encontramos um norte a ser seguido, e em outra foi respondida já que a idéia foi propor um novo propósito e foi entregue, porém não sabemos ao certo se foi respondida já que a idéia é fornecer um material para estudos e uma idéia de modelo organizacional. E pessoas da área de tecnologia precisam saber da existência desse material e aplicá-lo na pratica. O primeiro passo foi dado que foi elaborar esse trabalho, digamos que temos um problema a menos para ser resolvido.

O intuito da metodologia era além de propor um novo modelo organizacional, centralizar em todo o trabalho as idéias de vários autores de obras que descrevem os padrões de melhorias no desenvolvimento de rotinas de sistemas legado dentro do conceito de engenharia de software, e mitigar possíveis falhas dos modelos de processos atuais, apresentando uma idéia de modelo onde seria possível ligá-lo e desligá-lo na etapa falha de um determinado modelo ou ciclo de vida, podendo assim desligá-lo e seguir com o modelo escolhido normalmente, o trabalho foi feito com base na literatura a onde a forma de coletados dados foi através do Google acadêmico e Google livros, utilizando os métodos de observação direta e analise de conteúdo, onde foi elaborado um questionário onde um grupo formado por trinta pessoas contribuiu com a pesquisa respondendo-o.

5.1 LIMITAÇÕES

Diante da metodologia proposta percebesse que o trabalho poderia ter sido realizado com uma coleta de dados com um número maior de pessoas da área de tecnologia, mas devido à limitação de tempo e recursos, só foi possível realizar com a quantidade limite proposta.

5.2 RECOMENDAÇÕES

Recomendo pesquisas futuras sobre o tema proposto, sugiro que alimentem e aprofundem os métodos aqui abordados, testem esse método apresentado nesse trabalho para outros modelos de processos, e utilizem casos de uso em outras linguagens de programação, levantem suas hipóteses e desenvolva um novo artigo cientifico.

Sugiro aprofundar a pesquisa para um pós doutorado com o seguinte tema de pesquisa: organização e desenvolvimento de rotinas administrativas a partir da engenharia de software: proposta de implementação de um software de gestão nas

prefeituras do interior do estado do Ceará.

REFERÊNCIAS

ARMELIN, Danylo Augusto; SILVA, Simone Cecília Pelegrini da; COLUCCI, Claudio. **Sistemas de informação gerencial**. Londrina: Kls, 2016. 240 p.

BARDIN, Laurence. **Análise de conteúdo**. São Paulo: Edições 70, 2011. 279 p.

BERSSANETI, Fernando Tobal; BOUER, Gregório. **Qualidade Conceitos e Aplicações em Produtos, Projetos e Processos**. 1 ed. São Paulo: Blucher, 2013. 192 p.

BERTALANFFY, Ludwig von. **Teoria geral dos sistemas: Fundamentos, desenvolvimento e aplicações**. 5 ed. Rio de Janeiro: Vozes, 2010. 52 p.

CAMPOS, Vicente Falconi. **Gerenciamento pelas diretrizes**. São Paulo: FALCONI, 2013. 270 p.

CHIAVENATO, Idalberto. **Introdução a teoria geral da administração**. Rio de Janeiro: Elsevier, 2004. 650 p.

CHIAVENATO, Idalberto. **Teoria geral da administração**: Abordagens prescritivas e normativas. 7 ed. São Paulo: Manole, v. 1, 2014. 448 p.

CHIZZOTTI, Antonio. **Pesquisa em ciências humanas e sociais**. 12 ed. São Paulo: Cortez, 2016. 208 p.

ENGHOLM, Hélio. **Engenharia de Software na Prática**. São Paulo: Novatec, 2010. 440 p.

FERNANDES, Lúcia. **Oracle 9i**: Para desenvolvedores curso Completo. Rio de Janeiro: Axcel Books, 2002. 1614 p.

FERREIRA, André Ribeiro. **Análise e Melhoria de Processos**. Brasília: Enap, 2016. 114 p.

FOWLER, Martin. **Refatoração**: Aperfeiçoando o projeto de código existente. 2 ed. São Paulo: Novatec, 2019. 477 p.

FRANCO, miguel. Engenharia de software: desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos. **Revista Acadêmica Online**. São Paulo, 2020. 84 p. Disponível em: https://www.revistaacademicaonline.com. Acesso em: 16 nov. 2020.

FREITAS, Romualdo Rubens de. **Análise e Projeto de Software**. Cuiabá: Rede e-Tec Brasil, 2015. 60 p.

FULGENCIO, Paulo Cesar. **Glossário Vade Mecum**. Rio de Janeiro: Mauadx, 2007. 678 p.

GAMMA, Erich et al. **Padrões de Projeto, Soluções reutilizáveis de software orientado a objetos**: Design Patterns. 1 ed. Porto Alegre: Bookman, 2000. 368 p.

GARCIA, Adriana Amadeu; ARAUJO, Luis Cesar G. D.; MARTINES, Simone. **Gestão de processos**: Melhores Resultados e Excelência Organizacional. 2 ed. São Paulo: Atlas, 2017. 200 p.

GIL, ANTONIO Carlos. **Como elaborar projetos de pesquisa**. 6 ed. São Paulo: Atlas, 2017. 188 p.

GONÇALVES, Eduardo. **PL/SQL**: Domine a linguagem do banco de dados Oracle. São Paulo: Casa do Código, 2015. 414 p.

GUEDES, Gilleanes T. A. **UML 2**: Uma Abordagem Prática. 3 ed. São Paulo: Novatec, 2018. 496 p.

HEUSER, Carlos Alberto. **Projeto de banco de dados**. 6 ed. Porto Alegre: Bookman, 2009. 280 p.

KOSCIANSKI, André; SOARES, Michel dos Santos. **Qualidade de Software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2 ed. São Paulo: Novatec, 2007. 395 p.

LOBATO, David Menezes *et al*. **Estratégia de empresas**. 9 ed. Rio de Janeiro: FGV, 2009. 175 p.

LONG, Johnny; GARDNER, Bill; BROWN, Justin. **Google Hacking para Pentest**. Novatec Editora, v. 2, f. 132, 2019. 264 p.

MARCONI, Mariana; LAKATOS, Eva. **Fundamentos de metodologia científica.**. 8 ed. São Paulo: Atlas, 2017. 200 p.

MARKS, Sikberto Renaldo. **Estrutura e processos organizacionais**. Ijuí: Unijuí, 2008. 144 p.

MARSHALL, Isnard Junior *et al*. **Gestão da qualidade**. 9 ed. Rio de Janeiro: FGV, 2008. 204 p.

MARSHALL, Isnard Junior *et al*. **Gestão da qualidade e processos**. Editora FGV, v. 3, 2015.

MOLINARI, Willian. **Desconstruindo a Web**: As tecnologias por trás de uma requisição. Editora Casa do Código, v. 3, f. 117, 2016. 234 p.

MOREIRA, Daniel Augusto. **Administração da produção e operações**. 1 ed. São Paulo: Saraiva, 2013. 165 p.

PALADINI, Edson Pacheco. **Gestão da qualidade**: teoria e prática. 2 ed. São Paulo: Atlas, 2010. 339 p.

PARETO, Eduardo. **Cenários de TI**. 1 ed. Rio de Janeiro: SESES, 2016. 113 p.

PRESSMAN, Roger S.. **Engenharia de software**: uma abordagem profissional. 7 ed. São Paulo: AMGH, 2011. 780 p.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software**: Uma abordagem profissional. 8 ed. Porto Alegre: AMGH Editora Ltda, 2016. 968 p.

PRICE, Jason. **Oracle Database 11g SQL**: Domine SQL e PL/SQL no Banco de Dados Oracle. 1 ed. Porto Alegre: Artmed, 2008. 684 p.

PRÉVE, Altamiro Damian; MORITZ, Gilberto de Oliveira; PEREIRA, Maurício Fernandes. **Organização, Processos e Tomada de Decisão**. Florianópolis: UFSC, 2010. 186 p.

PÁDUA, Wilson. Engenharia de Software: Fundamentos, Métodos e Padrões. 3. ed. Rio de Janeiro: LTC, 2012. 1358 p.

SILVA, Airton Marques. **Metodologia da Pesquisa**. 2 ed. Fortaleza: EDUECE, 2015. 110 p.

SILVA, Andressa Hennig; FOSSÁ, Andressa Hennig. **Análise de conteúdo**: exemplo de aplicação da técnica para análise de dados qualitativos. Qualitas Revista Eletrônica. Campina Grande, 2015. 14 p. Disponível em: http://arquivo.revista.uepb.edu.br/index.php/qualitas/article/view/2113/1403. Acesso em: 5 set. 2020.

SILVA, Andressa Hennig; FOSSÁ, Maria Ivete Trevisan. Análise de conteúdo: exemplo de aplicação da técnica para análise de dados qualitativos. **Qualitas Revista Eletrônica**, Campina Grande. 14 p, 2015. Disponível em: http://revista.uepb.edu.br/index.php/qualitas. Acesso em: 15 nov. 2021.

SLACK, Nigel; CHAMBERS, Stuart ; JOHNSTON, Robert. **Administração da Produção**. 3 ed. São Paulo: Atlas, 2009. 728 p.

SOMMERVILLE, ian. **Engenharia de software**. 10 ed. São Paulo: Pearson Education, 2016. 811 p.

SOMMERVILLE, Ian. **Engenharia de software**. 9 ed. São Paulo: Pearson Education, 2011. 544 p.

VENTURA, Plínio. **Requisitos de Software**: Uma visão detalhada sobre Requisitos Funcionais, Requisitos Não-Funcionais e Regras de Negócio. Belo Horizonte: Indtech, 2016. 57 p.

GLOSSÁRIO

V & V	Verificação e validação

ANEXO A — QUESTIONÁRIO

Solicitamos a sua colaboração e sinceridade na participação desse questionário que se enquadra numa investigação no âmbito do Doutorado em engenharia de software cujo tema de pesquisa é, SIMPLES SEQUENCIA PARA CONTROLAR E ORGANIZAR O DESENVOLVIMENTO DE ROTINAS ADMINISTRATIVAS COM ENGENHARIA DE SOFTWARE: PROPOSTA DE IMPLEMENTAÇÃO NO SETOR DE REGISTRO HOSPITALAR, da Universidade Selinus. Pretende-se com esta entrevista colher sua experiência e opinião relativamente ao contributo que dá para o desenvolvimento deste trabalho. Este questionário é de natureza anônima e todas as informações recolhidas são estritamente confidenciais, pelo que nos comprometemos a fazer uso da informação recolhida, exclusivamente, para fins a que se destina a investigação.

Desde já agradecemos pelo seu precioso tempo concedido e pela sua colaboração.

1) Contudo o modelos de processo de software, segundo (SOMMERVILLE, 2016, p. 19), “um modelo de processo de software é uma representação simplificada de um processo de software. Cada modelo representa uma perspectiva particular de um processo e, portanto, fornece informações parciais sobre ele”. Certamente talvez seria interessante concordar com essa citação sobre modelos de processos de software? Sim ou não?

2) Portanto (SOMMERVILLE, 2016, p. 19) “esses modelos genéricos não são descrições definitivas dos processos de software. Pelo contrário, são abstrações que podem ser usadas para explicar diferentes abordagens de desenvolvimento de software”. Sobretudo como profissional, seria contudo interessante uma nova abordagem para auxiliá-lo no processo de melhoria de rotinas do software da organização? Sim ou não?

3) Segundo (SOMMERVILLE, 2016, p. 19) faz uma abordagem a três modelos de processos, o “modelo em cascata (especificação de requisitos, projeto de software, implementação, teste), desenvolvimento incremental (O sistema é desenvolvido como uma série de versões (incrementos), de maneira que cada versão adiciona funcionalidade à anterior), engenharia de software orientada a reuso (O processo de desenvolvimento do sistema concentra-se na integração desses componentes em um sistema já existente em vez de desenvolver um sistema a partir

do zero)”. Conclui-se que seria interessante implementar um conceito de organização de código fonte levando em conta todos esses modelos já existentes? Sim ou não?

4) É da área de engenharia de software ou de outras relacionadas ao desenvolvimento de software? Responda sim ou não.

5) Seria formando na área de engenharia de software ou de outras relacionadas ao desenvolvimento de software? Responda sim ou não.

6) Teria 2, 5 ou mais anos de experiência profissional na área de desenvolvimento de software ou áreas relacionadas? Responda sim ou não.

7) Seria maior de 18 anos? Responda sim ou não.

8) Saberia fazer de certa forma, levantamento de requisitos ou testes de software ou reuso de código fonte ou alguma etapas de processo de implementação de um software? Sim ou não?

9) Vivenciou sim ou não, profissionalmente algum desses modelos citados no questionário? Responda sim ou não.

10) De certa forma a vivência profissional e experiência no ramo de atividade em questão, um modelo de processo ou ciclo de vida de um software já apresentou falhas em seus processos e atividades de desenvolvimento, operação e manutenção? Sim ou não?

Obrigado.