Universidad Europea del Atlántico

ÁREA DE TECNOLOGIAS TIC

TÍTULO DA DISSERTAÇÃO

Engenharia de software: desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos

Dissertação para a obtenção do grau de:

Mestrado em Direção Estratégica em Engenharia de Software

Apresentado por:

Miguel Marcelo Nascimento Franco

BRMDEISW2759383

Orientador:

Dr. Roberto Fabiano Fernandes

FORTALEZA, BRASIL

2020

AGRADECIMENTOS

Aos professores das disciplinas e comitê de ética da universidade pelas orientações, e ao meu orientador. Deixo aqui meus mais sinceros agradecimentos, obrigado a todos.

COMPROMISSO DO AUTOR

Eu, Miguel Marcelo Nascimento Franco, portador do documento de matricula BRMDEISW2759383 e aluno do programa acadêmico Mestrado em Direção Estratégica em Engenharia de Software, declaro que:

O conteúdo do presente documento é um reflexo do meu trabalho pessoal e manifesto que, diante de qualquer notificação de plágio, cópia ou prejuízo à fonte original, sou responsável direto legal, financeira e administrativamente, sem afetar o Orientador do trabalho, a Universidade e as demais instituições que colaboraram neste trabalho, assumindo as consequências derivadas de tais práticas.

Assinatura: 

Fortaleza, 26 de março de 2020

Att: Direção Acadêmica

Venho por meio desta, autorizar a publicação eletrônica da versão aprovada de minha dissertação com título: Engenharia de software: Desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos, na revista cientifica online e virtual e em outras mídias de divulgação eletrônica desta.

Informo abaixo os dados para descrição do trabalho:

Título	**Engenharia de software: desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos**
Autor	**Miguel Marcelo Nascimento Franco; Roberto Fabiano Fernandes**
Resumo	**A pesquisa analisa o desenvolvimento das rotinas em PL/SQL dentro das organizações do Brasil e identifica a necessidade do desenvolvimento estratégico centralizado para manter e agilizar os processos como um todo das organizações, tendo em conta a importância de manter, o estado de manutenção as rotinas do software, empresarial e seus benefícios.**
Programa	**Mestrado em Direção Estratégica em Engenharia de Software**
Palavras-chave	**Linguagem Programação; Software Organizacional; PL/SQL; Direção Estratégica; Engenharia software**
Contato	**guelmi_marcelo@hotmail.com**

Atenciosamente,

Assinatura: 

ÍNDICE GERAL

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMO

A engenharia de software tem um papel fundamental no desenvolvimento das rotinas internas das organizações. No entanto, centralizá-las estrategicamente contribui, agiliza e mantém os processos, regra de negócio da organização. Pretendemos com este trabalho analisar o desenvolvimento dessas rotinas dentro das organizações do Brasil. Tem como objetivo geral analisar o desenvolvimento dessas rotinas dentro das organizações do Brasil. Para o efeito, vamos fazer uma pesquisa exploratória com recurso bibliográfico a grandes obras de autores renomados com horárias da área de pesquisa, análise documental para obtenção dos dados. A abordagem qualitativa será usada para efeitos de análise e interpretação de dados. O referencial teórico apresenta diferentes abordagens de desenvolvimento. Esperamos que o presente trabalho sirva de base para futuros estudos e apresente as principais características do desenvolvimento estratégico da organização buscando centralizar tais rotinas bem como manter e agilizar as regras de negócio das organizações do Brasil. A conclusão do estudo permite considerar necessário o desenvolvimento estratégico centralizado para manter e agilizar os processos como um todo das organizações, tendo em conta a importância de manter, o estado de manutenção as rotinas do software, empresarial e seus benefícios.

Palavras-chave: Linguagem e programação; Software organizacional; PL/SQL; Direção Estratégica; Engenharia software.

ABSTRACT

Software engineering plays a key role in the development of organizations' internal routines. However, centralizing them strategically contributes, streamlines and maintains the processes, the organization's business rule. With this work we intend to analyze the development of these routines within organizations in Brazil. Its general objective is to analyze the development of these routines within organizations in Brazil. For this purpose, we are going to do an exploratory research with a bibliographic resource of great works by renowned authors with hours in the research area, document analysis to obtain the data. The qualitative approach will be used for the purposes of data analysis and interpretation. The theoretical framework presents different development approaches. We hope that the present work will serve as a basis for future studies and present the main characteristics of the strategic development of the organization, seeking to centralize such routines as well as maintain and streamline the business rules of organizations in Brazil. The conclusion of the study allows considering the centralized strategic development necessary to maintain and streamline the processes as a whole of the organizations, taking into account the importance of maintaining, the state of maintenance, the software, business routines and their benefits.

Keywords: Programming and language; Organizational software; PL/SQL; Strategic Direction; Software engineering]

INTRODUÇÃO

Acredita-se que as organizações possuem softwares com códigos desestruturados, porém, nada impede que sejam estruturados, também conhecidos como “sistema legado”, que podem ser muito antigos.

> 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 e Maxim, 2016, p. 8).

Esses processos de melhorias avançam em larga escala e geram desconforto na hora de adaptá-los e ajustá-los internamente. Referindo-se ao desenvolvimento de rotinas, nossa área de pesquisa, muitos dos códigos nesses sistemas organizacionais apresentam um difícil entendimento, testes e resultados sem documentação tornando difícil a manutenção de suas rotinas.

> Infelizmente, de vez em quando uma característica adicional está presente em software legado – a baixa qualidade. Ás vezes, os sistemas legados têm projetos inextensíveis, códigos de difícil entendimento, documentação deficiente ou inexistente, casos de teste e resultados que nunca foram documentados, um histórico de alterações mal gerenciado – a lista pode ser bastante longa. Ainda assim, esses sistemas dão suporte a funções vitais de negócio e são indispensáveis para ele (Pressman e Maxim, 2016, p. 8).

Embora os processos de melhorias desses softwares, devido à sua baixa qualidade, comprometam e tornem a manutenção de suas velhas e novas rotinas um pouco lenta.

> Esses sistemas evoluem devido a uma ou mais dessas razões, o software deve ser adaptado para atender ás necessidades de novos ambientes ou de novas tecnologias computacionais, o software deve ser aperfeiçoado para implementar novos requisitos de negócio. Quando essas modalidades de evolução ocorrem, um sistema legado deve passar por reengenharia para que permaneça viável no futuro (Pressman e Maxim, 2016, p. 8).

Contudo, ressalto aqui a relevância do tema, engenharia de software, desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos.

1.1. Tema/Área de pesquisa e contextualização

O interesse pelo estudo relacionado à engenharia de software, desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos, tem crescido a nível mundial. Acerca do software e desenvolvimento das rotinas, para Sommerville (2011, p. 9) 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, referencial citado. Contudo, a uma relevância para o campo de estudo.

Ressaltando os pontos estratégicos e relevantes para a área de pesquisa e conclusão, no ambiente de desenvolvimento de software, para Sommerville (2011, p. 51) “estabelece-se confiança entre clientes e desenvolvedores e cria-se uma cultura positiva, na qual todo mundo espera que o projeto tenha êxito”. Contudo, a entrega das rotinas para agilizar os processos contidos como um todo no projeto, precisa desse laço de confiança entre desenvolvedores e clientes.

Continuando e contextualizando, outro ponto é centralizar as rotinas e sub-rotinas do software. No contexto, centralizar é organizar? Por que devemos organizar essas rotinas no software? Qual é a relevância da centralização? Não podemos responder a essas perguntas sem antes ler o que diz Sommerville (2011, p. 398): “a separação de interesses é um princípio-chave de projeto e implementação de software”. O que isso quer nos dizer? Isso significa que você deve organizar seu software de tal forma que cada elemento do programa (classe, método, procedimento etc.) faça apenas uma coisa. Assim poderá concentrar-se nesse elemento sem se preocupar com outros elementos no programa.

E o entendimento do código fonte do software? Que além de inúmeras linhas, torna a leitura e releitura demorada. Sobretudo, para Sommerville (2011, p. 398), “poderá entender cada parte do programa conhecendo seus interesses, sem a necessidade de entender outros elementos. Quando as mudanças forem necessárias, elas serão feitas em um número pequeno de elementos”.

Em síntese, o tema proposto, engenharia de software, desenvolver rotinas em PL/SQL, centralizadas estratégias para agilizar e manter processos, adequa-se à área de tecnologia.

1.2. Problema de pesquisa

Para melhor compreensão do estudo, os procedimentos metodológicos caracterizam-se por uma revisão bibliográfica, de cunho exploratório descritivo, retomando-se o objeto de estudo, ou seja, engenharia de software, desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos. 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, descobrir 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: Desenvolver rotinas em PL/SQL, centralizadas estrategicamente, agiliza e mantém processos?

1.2.1. Justificativa/motivação para resolver

O presente trabalho se justifica 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.

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, motivou a realização do presente estudo.

1.3. Objetivo geral

Analisar o desenvolvimento de rotinas, centralizadas estrategicamente para agilizar e manter processos.

1.4. Objetivos específicos

i) Descrever as características de desenvolvimento de rotinas de software;
ii) Identificar as maiores dificuldades de manutenção das rotinas do sistema legado;
iii) Apresentar pontos relevantes de pesquisas para processos de engenharia de software;
iv) Responder como organizar rotinas com base na pesquisa de engenharia de software;
v) Averiguar até que ponto o desenvolvimento é essencial para o software organizacional.

1.5. Resultados esperados

Pretendemos com esta pesquisa apresentar um perfil da engenharia de software, desenvolvendo rotinas em PL/SQL, centralizadas estrategicamente para agilizar e manter processos. Esperamos, no final, fornecer dados empíricos que possibilitem aprofundar mais esta temática.

1.6. Estrutura da dissertação

Esta dissertação está estruturada em cinco capítulos. 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 da dissertação. 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. REVISÃO DA LITERATURA

O referencial teórico consiste em mostrar possibilidades de organização a estrutura desenvolvida do sistema, centralizando, organizando, analisando, desenvolvendo ou criando rotinas para o software legado, chegando ao processo de melhoria e adaptação como um todo. Contudo, esse embasamento contribui para formar um ponto de vista e ao final deve-se mostrar uma opinião formada nessa base. Sobretudo, a centralização de rotinas pode agilizar e manter processos de um sistema legado? Iremos fundamentar possíveis razões que possam levar à essas possibilidades.

De acordo com Córdova (2020), a engenharia de software, foi criada a partir da intersecção de várias disciplinas, uma vez que não é ciência básica explicativa, e tendo como motivador de sua criação, a funcionalidade relacionada à resolução de problemas.

No entanto, Sommerville (2011) ressalta que:

> O nome engenharia de software foi proposto em 1969, na conferência da OTAN, para a discussão de problemas relacionados com desenvolvimento de software. Grandes softwares atrasavam, não entregavam a funcionalidade de que os usuários necessitavam, custavam mais do que o esperado e não eram confiáveis.

Para Pádua (2000), “a tecnologia só resolve problemas quando é usada por pessoas qualificadas, dentro de processos adequados. Os sistemas de informática são os produtos da tecnologia de tratamento da informação”.

Contudo, temos três sínteses sobre problemas relacionados ao desenvolvimento para a engenharia de software, discussão e funcionalidade de resolução de problemas e pessoas qualificadas. Será que centralizar rotinas ajudará a agilizar e manter os processos de um software legado? Será que a centralização dessas rotinas resolverá algum desses problemas do software, ou ao menos organizar sua estrutura de código e leitura? Essa é uma questão a se pensar ao longo do nosso referencial.

2.1. Conceito de desenvolvimento na engenharia

Falar em desenvolvimento é pensar em processo? Para Lobo (2009) “quando se fala em desenvolvimento de software, deve-se pensar em processo. Um processo, ou método, é definido para se obter software cada vez mais rápido, de forma controlada e organizada”.

Subentende-se que a definição de desenvolver as rotinas de um software, pode ser agilizada para se ter um software mais rápido e, consequentemente, manter seus processos, de maneira controlada, centralizada e organizada estrategicamente.

Vale ressaltar que, para Pádua (2000, p. 12):

> A engenharia de software se preocupa com o software enquanto produto. Estão fora de seu escopo programas que são feitos unicamente para diversão do programador. Estão fora de seu escopo também pequenos programas descartáveis, feitos por alguém exclusivamente como meio para resolver um problema, e que não serão utilizados por outros.

Entretanto, para Sommerville (2011, p. 3):

> A engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software, mais do que a programação individual. Ela inclui técnicas que apoiam especificação, projeto e evolução de programas, que normalmente não são relevantes para o desenvolvimento de software pessoal.

Para Pressman (2011, p. 70):

> Independentemente do processo de software escolhido, os desenvolvedores de software complexos, invariavelmente, implementam um conjunto de recursos, funções e conteúdo localizados. Essas características de software localizadas são modeladas como componentes (por exemplo, classes orientadas a objetos) e, em seguida, construídas dentro do contexto da arquitetura do sistema. À medida que os modernos sistemas baseados em computadores se tornam mais sofisticados (e complexos), certas restrições propriedades exigidas pelo cliente ou áreas de interesse técnico se estendem por toda a arquitetura. Algumas restrições são propriedades de alto nível de um sistema (por exemplo, segurança, tolerância a

falhas). Outras afetam funções (por exemplo, a aplicação de regras de negócio), sendo que outras são sistêmicas (por exemplo, sincronização de tarefas ou gerenciamento de memória).

Ainda de acordo com Pressman (2011, p. 40):

> A base para a engenharia de software é a camada de processos. O processo de engenharia de software é a liga que mantém as camadas de tecnologia coesas e possibilita o desenvolvimento de software de forma racional e dentro do prazo. O processo define uma metodologia que deve ser estabelecida para a entrega efetiva de tecnologia de engenharia de software. O processo de software constitui a base para o controle do gerenciamento de projetos de software e estabelece o contexto no qual são aplicados métodos técnicos, são produzidos produtos derivados (modelos, documentos, dados, relatórios, formulários etc.), são estabelecidos marcos, a qualidade é garantida e mudanças são geridas de forma apropriada. Os métodos da engenharia de software fornecem as informações técnicas para desenvolver softwares. Os métodos envolvem uma ampla gama de tarefas, que incluem: comunicação, análise de requisitos, modelagem de projeto, construção de programa, testes e suporte. Os métodos da engenharia de software baseiam-se em um conjunto de princípios básicos que governam cada área da tecnologia e inclui atividades de modelagem e outras técnicas descritivas. As ferramentas da engenharia de software fornecem suporte automatizado ou semiautomatizado para o processo e para os métodos. Quando as ferramentas são integradas, de modo que as informações criadas por uma ferramenta possam ser usadas por outra, é estabelecido um sistema para o suporte ao desenvolvimento de software, denominado engenharia de software com o auxílio do computador.

Segundo Engholm (2010, p. 30):

> A manutenção de software pode estar associada a motivos diferentes, originando diferentes tipos de manutenção. Entre estes podemos listar:
> • Corretiva: modificação do software com o objetivo de corrigir falhas.
> • Evolutiva: relacionada à inclusão de novas funcionalidades ou ao desempenho.
> • Adaptativa: relacionada à adaptação do software a ambientes operacionais diferentes.

Com base nos vários tipos de manutenção, podemos dizer que as manutenções estão associadas a pelo menos um dos itens a seguir:

• Remoção de defeitos.
• Adição de funcionalidades no sistema.
• Alteração de funcionalidades.
• Evolução das funcionalidades.
• Ajustes.
• Atualização.
• Melhora de desempenho.

Para Sommerville (2011, p. 3):

> Inúmeras pessoas escrevem programas. Pessoas envolvidas com negócios, escrevem programas em planilhas para simplificar seu trabalho, cientistas e engenheiros escrevem programas para processar seus dados experimentais; e há aqueles que escrevem programas como hobby, para seu próprio interesse e diversão. No entanto, a maior parte do desenvolvimento de software é uma atividade profissional, em que o software é desenvolvido para um propósito específico de negócio, para inclusão em outros dispositivos ou como produtos de software como sistemas de informação, sistemas CAD etc" O software profissional, o que é usado por alguém além do seu desenvolvedor, é normalmente criado por equipes, em vez de indivíduos. Ele é mantido e alterado durante sua vida.
>
> A engenharia de software tem por objetivo apoiar o desenvolvimento profissional de software, mais do que a programação individual. Ela inclui técnicas que apoiam especificação, projeto e evolução de programas, que normalmente não são relevantes para o desenvolvimento de software pessoal.
>
> Muitas pessoas pensam que software é simplesmente outra palavra para programas de computador. No entanto, quando falamos de engenharia de software, não se trata apenas do programa em si, mas de toda a documentação associada e dados de configurações necessários para fazer esse programa operar corretamente. Um sistema de software desenvolvido profissionalmente é, com frequência, mais do que apenas um programa: ele normalmente consiste em uma série de programas separados e arquivos de configuração que são usados para configurar esses programas. Isso pode incluir documentação do sistema, que descreve a sua estrutura, documentação do usuário, que explica como usar o sistema, e sites, para usuários baixarem a informação recente do produto.

Essa é uma diferença importante entre desenvolvimento de software profissional e amador. Se você está escrevendo um programa para si mesmo, que ninguém mais usará, você não precisa se preocupar em escrever o manual do programa, documentar sua arquitetura etc. No entanto, se você está escrevendo um software que outras pessoas usarão e no qual outros engenheiros farão alterações, então você provavelmente deve fornecer informação adicional, assim como o código do programa.

Engenheiros de software se preocupam em desenvolver produtos de software (ou seja, software que pode ser vendido para um cliente). Existem dos tipos de produtos de software:

1. Produtos genéricos. Existem sistemas *stand-alone*, produzidos por uma organização de desenvolvimento e vendidos no mercado para qualquer cliente que esteja interessado em comprá-los. Exemplos desse tipo de produto incluem softwares para PCs, como ferramentas de banco de dados, processadores de texto, pacotes gráficos e gerenciamento de projetos. Também incluem as chamadas aplicações verticais projetadas para um propósito específico, como sistemas de informação de bibliotecas, sistemas de contabilidade ou sistemas de manutenção de registros odontológicos.
2. Produtos sob encomenda. Estes são os sistemas encomendados por um cliente em particular. Uma empresa de software desenvolve o software especialmente para esse cliente. Exemplos desse tipo de software são sistemas de controle de dispositivos eletrônicos, sistemas escritos para apoiar um processo de negócio específico e sistemas de controle de tráfego aéreo.

2.2. Introdução à linguagem PL/SQL

Abordando um ponto introdutório à linguagem de desenvolvimento, de acordo com Gonçalves (2015, p. 2) “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, Prince (2009, p. 368), 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”.

Fernandes (2002, p. 249), ressalta 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. E para o referencial proposto, no desenvolvimento de rotinas? Segundo Fernandes (2002, p. 248) “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 estratégia para agilizar e manter os 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”.

Prince (2009), ressalta que “os programas em PL/SQL são divididos em estruturas conhecidas como blocos, com cada bloco contendo instruções PL/SQL e SQL”, para 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”.

Segundo Fernandes (2002, p. 250) a estrutura de um bloco PL/SQL é composta de três partes:

> Uma parte declarativa, onde definimos as variáveis locais àquele bloco. Uma parte de lógica, onde definimos a ação que aquele bloco deve realizar, incluindo a declaração de outros blocos subordinados (ou embutidos) a este. Uma parte de tratamento de erros, que permite que tenhamos acesso ao erro ocorrido e a determinação de uma ação de correção. Nessa parte, também podemos declarar outros blocos subordinados.

Para Gonçalves (2015, p. 8):

> 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 pela aplicação e também a expressão *exception*, que delimita uma área para tratamento de erros. 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.

Segundo Gonçalves (2015, p. 5):

> Quando temos dentro da aplicação comandos SQL distintos, eles são enviados um a um para o servidor do banco de dados. Dessa forma, a aplicação envia um comando para o servidor, espera a resposta e depois envia outro.
> Quando temos blocos PL/SQL, eles são enviados por completo ao servidor do banco de dados, não importando o seu teor. Dentro deste bloco podemos ter vários comandos SQL e demais estruturas em PL/SQL. Desse modo economizamos tempo, pois a aplicação envia de uma só vez todas as solicitações, e o número de respostas esperadas também reduz muito. 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.

Para Prince (2009, p. 405):

> Por padrão, cada unidade de programa PL/SQL é compilada em código legível pela máquina, na forma intermediária. Esse código legível pela máquina é armazenado no banco de dados e interpretado sempre que o código é executado. Com a compilação nativa PL/SQL, o PL/SQL é transformado em código nativo e armazenado em bibliotecas compartilhadas. O código nativo é executado muito mais rapidamente do que o código intermediário, pois não precisa ser interpretado antes da execução.

Segundo Gonçalves (2015, p. 7):

> A linguagem PL/SQL trabalha em blocos de comando. Dentro de bloco podemos ter outros blocos, que neste caso são chamados de sub-blocos. 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.

Ainda de acordo com Gonçalves (2015, p. 9),

> A programação em bloco torna os programas mais estruturados e limpos. Principalmente, quando utilizamos vários sub-blocos para a realização de determinadas tarefas distintas, como a seleção de dados, uma atualização ou exclusão. 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.

Para Price (2009), “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, p. 12),

> Para construir um programa em PL/SQL temos que trabalhar em nível de bloco. 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.
> 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, além do uso de operadores relacionais e numéricos.

Para Price (2009), “os tipos PL/SQL são semelhantes aos tipos de coluna de banco de dados”.

2.3. Qualidade no desenvolvimento de rotina

Para Pádua (2000, p. 17):

> Geralmente a qualidade de um produto decorre diretamente da qualidade do processo utilizado na produção dele. Note-se que importa aqui a qualidade do processo efetivamente utilizado, não do ‘processo oficial’ que pode eventualmente estar descrito nos manuais da organização. Muitas vezes os processos oficiais não são seguidos na prática, por deficiência de ferramentas, 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 (2011, p. 455):

> Às vezes, garantia de qualidade significa simplesmente a definição de procedimentos, processos e padrões que visam reforçar que a qualidade de software seja atingida. Em outros casos, a garantia de qualidade também inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicadas após o produto ter sido entregue por uma equipe de desenvolvimento.

Segundo Pressman (2011, p. 3):

> Apesar de gerentes e profissionais envolvidos com a área técnica reconhecerem a necessidade de uma abordagem mais disciplinada em relação ao software, eles continuam a discutir a maneira como essa disciplina deve ser aplicada. Muitos indivíduos e empresas desenvolvem software de forma desordenada, mesmo ao construírem sistemas dirigidos às mais avançadas tecnologias. Grande parte de profissionais e estudantes não estão cientes dos métodos modernos. E, como consequência, a qualidade do software que produzem é afetada. Além disso, a discussão e a controvérsia sobre a real natureza da abordagem de engenharia de software continuam. 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 e Soares (2007, p. 42) “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 e Soares (2007, p. 30), 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”. 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:
> • tamanho e complexidade do software sendo construído;
> • número de pessoas envolvidas no projeto;
> • ferramentas utilizadas;

• custos associados à existência de erros;
• custos associados à detecção e à remoção de erros.

Segundo Koscianski e 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, p. 20):

> A qualidade contempla uma série de objetivos da construção de software, conhecidos como requisitos não-funcionais, tais como extensibilidade, capacidade de manutenção, reutilização de código, desempenho, escalabilidade, usabilidade e confiabilidade nos dados apresentados pela aplicação. Podemos presenciar uma série de problemas enfrentados por empresas que investem no desenvolvimento de sistemas sem a utilização de engenharia de software, 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, p. 20):

> Para clarificar esses problemas, são apresentadas a seguir as consequências práticas de desenvolver softwares de modo ad hoc, sem utilização de processos definido, orientação a objetos e melhores práticas:
> • Softwares difíceis de se dar manutenção, tanto corretiva quanto evolutiva.
> • Softwares difíceis de se implementar alterações, tanto corretivas quanto evolutivas.
> • Reutilização de código mal elaborado e sujeito a geração/propagação de erros em outras partes do sistema em desenvolvimento.
> • Sistemas com baixo desempenho e escalabilidade inadequada.
> • Baixa eficiência no desenvolvimento, com analistas desenvolvendo as mesmas funcionalidades diversas vezes.
> • Falta de confiança nos dados apresentados pelo sistema, fazendo com que usuários deixem de utilizar o sistema por não confiar nas informações apresentadas.

• Baixa qualidade de código.

Em contrapartida aos efeitos do desenvolvimento ad hoc, 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. Esse desejo faz com que muitas empresas prefiram não utilizar processos de engenharia de software, pensando na falsa economia de tempo e diminuição de custos. Que essa ideia 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. 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, p. 83), “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, p. 4):

> 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, p. 7):

> 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:
>
> 1. Aplicações *stand-alone*. Essas são as aplicações executadas em um computador local, como um PC. Elas contêm toda a funcionalidade necessária e não precisam estar conectadas a uma rede. Exemplos de tais aplicações são

aplicativos de escritório em um PC, programas CAD, software de manipulação de fotos etc.

2. Aplicações interativas baseadas em transações. São aplicações que executam em um computador remoto, acessadas pelos usuários a partir de seus computadores ou terminais. 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. Essa classe de aplicações também inclui sistemas corporativos, em que uma empresa fornece acesso a seus sistemas através de um navegador Web ou um programa cliente especial e serviços baseados em nuvem, como é o caso de serviços de e-mail e compartilhamento de fotos. Aplicações interativas frequentemente incorporam um grande armazenamento de dados, que é acessado e atualizado em cada transação.

3. Sistemas de controle embutidos. São sistemas de controle que controlam e gerenciam dispositivos de hardware. Numericamente, é provável que haja mais sistemas embutidos do que de qualquer outro tipo. 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.

4. Sistemas de processamento de lotes. São sistemas corporativos projetados para processar dados em grandes lotes. 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.

5. Sistemas de entretenimento. São sistemas cuja utilização principal é pessoal e cujo objetivo é entreter o usuário. A maioria desses sistemas é de jogos de diferentes tipos. A qualidade de interação com o usuário é a característica particular mais importante dos sistemas de entretenimento.

6. Sistemas para modelagem e simulação. São sistemas que incluem vários objetos separados que interagem entre si, desenvolvidos por cientistas e engenheiros para modelar processos ou situações físicas. Esses sistemas geralmente fazem uso intensivo de recursos computacionais e requerem sistemas paralelos de alto desempenho para executar.

7. Sistemas de coleta de dados. São sistemas que coletam dados de seu ambiente com um conjunto de sensores e enviam esses dados para outros sistemas para processamento. O software precisa interagir com sensores e

frequentemente é instalado em um ambiente hostil, por exemplo, dentro de uma máquina ou em um lugar remoto.

8. Sistemas de sistemas. São sistemas compostos de uma série de outros sistemas de software. Alguns deles podem ser produtos genéricos de software, como um programa de planilha eletrônica. Outros sistemas do conjunto podem ser escritos especialmente para esse ambiente.

Koscianski e Soares (2007) fazem a seguinte pergunta: “quais são os problemas enfrentados na construção e utilização de software?” Nas palavras de Koscianski e Soares (2007, p. 22), há uma pequena lista:

- cronogramas não observados;
- projetos com tantas dificuldades que são abandonados;
- módulos que não operam corretamente quando combinados;
- programas que não fazem exatamente o que era esperado;
- programas tão difíceis de usar que são descartados;
- programas que simplesmente param de funcionar.

Para Pádua (2000, p. 17):

> Geralmente a qualidade de um produto decorre diretamente da qualidade do processo utilizado na produção dele. Note-se que importa aqui a qualidade do processo efetivamente utilizado, não do "processo oficial", que pode eventualmente estar descrito nos manuais da organização. Muitas vezes os processos oficiais não são seguidos na prática, por deficiência de ferramentas, por falta de qualificação das pessoas, ou porque pressões de prazo levam os gerentes dos projetos a eliminar etapas relacionadas com controle da qualidade.
>
> Em um produto de software de má qualidade, muitos requisitos não são atendidos completamente. As deficiências de conformidade com os requisitos se manifestam de várias maneiras. Em alguns casos, certas funções não são executadas corretamente sob certas condições, ou para certos valores de entradas. Em outros casos, o produto tem desempenho insuficiente, ou é difícil de usar.
>
> Cada requisito não atendido é um defeito. No mundo informático, criou-se a usança de chamar de “bugs” os defeitos de software. Assim, erros técnicos adquirem conotação menos negativa, que lembra simpáticos insetos de desenho

> animado. E 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 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. Eles levam a desentendimentos sem solução entre o fornecedor e o cliente do produto. 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.

Ainda segundo Pádua (2000, p. 18):

> Em todas as fases do desenvolvimento de software as pessoas introduzem defeitos. Eles decorrem de limitações humanas: erros lógicos, erros de interpretação, desconhecimento de técnicas, falta de atenção, ou falta de motivação. Em todo bom processo existem atividades de garantia da qualidade, tais como revisões, testes e auditorias. Estas atividades removem parte dos defeitos introduzidos.
>
> 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. Quanto mais tarde um defeito é corrigido, mais cara é a sua correção, por várias razões que serão discutidas posteriormente. O pior caso acontece quando o defeito chega ao produto final. Neste caso, ele só será removido através de uma operação de manutenção. 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

2.4. A evolução dos sistemas legados

Segundo Pressman e Maxim (2016, p. 8) “programas mais antigos frequentemente denominados softwares legado têm sido foco de contínua atenção e preocupação”, portanto, “um software legado é caracterizado pela longevidade e criticidade de negócio”.

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. Ele representa um legado. O importante é que ainda está em operação e desempenha funções vitais da empresa”. 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, p. 36):

> Infelizmente, algumas vezes, há uma característica adicional que pode estar presente em um software legado: baixa qualidade. 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. 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, p. 46)

> Muitos sistemas foram desenvolvidos para grandes empresas: sistemas caros, complexos, com banco de dados da época, desenvolvidos para os computadores mainframes. Só que estes sistemas são extremamente estáveis e atendem até às demandas atuais. 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. Estes sistemas fornecem serviços essenciais, mas têm difícil manutenção. Se procurarmos uma definição formal do termo sistema legado, certamente será difícil de encontrar. Por isso serão apresentadas suas características:

• 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.

• Outra característica é que eles funcionam em hardwares obsoletos, principalmente nos mainframes, cujos componentes são extremamente caros e raros. Estes sistemas utilizam linguagens como COBOL, banco de dados e formatos de arquivos obsoletos.

• 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. Geralmente, os profissionais que desenvolveram estes sistemas já se aposentaram ou estão em processo de aposentadoria.

• A 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 (2009, p. 28):

> 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, por 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 encontram mais na empresa.
>
> Esse tipo de código é conhecido como “código alienígena” ou “software legado”. 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. Além disso, nenhum dos profissionais da equipe atual trabalhou em seu projeto inicial e, para piorar, 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.

2.5. Levantamento de requisito para o desenvolvimento de rotinas

Especificações, documentação e pessoas, são essenciais para o desenvolvimento de rotinas do sistema legado da organização. Segundo Koscianski e Soares (2007, p. 28):

> Os requisitos foram especificados por uma pessoa: logo, é preciso saber claramente do que ela precisa. 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.

Ressaltando, para Sommerville (2011, p. 16) “as ideias 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.

Ainda segundo Sommerville (2011, p. 24):

> 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. 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.
>
> 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, p. 127):

> O amplo espectro de tarefas e técnicas que levam a um entendimento dos requisitos é denominado engenharia de requisitos. 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. Ela deve ser adaptada às necessidades do processo, do projeto, do produto e das pessoas que estão realizando o trabalho. A engenharia de requisitos constrói uma ponte para o projeto e para a construção.

> A engenharia de requisitos constrói uma ponte para o projeto e para a construção. Mas onde começa essa ponte? Alguém pode argumentar que ela começa na base dos interessados no projeto (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. 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. 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; as necessidades específicas que o projeto e a construção devem atender; 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.
>
> 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, especificando a solução sem ambiguidades validando a especificação e gerenciando as necessidades à medida que são transformadas em um sistema operacional. Ela abrange sete tarefas distintas: concepção, levantamento, elaboração, negociação, especificação, validação e gestão. É importante notar que algumas delas ocorrem em paralelo e todas são adaptadas às necessidades do projeto.

Sobre o planejamento no levantamento de requisitos Pressman (2011, p. 88) ressalta:

> 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 XP, 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.
>
> A atividade de ‘ouvir’ conduz à criação de um conjunto de ‘histórias’ (também denominado histórias de usuários) que descreve o resultado, as características e a funcionalidade requisitados para o software a ser construído. Cada história é escrita pelo cliente e é colocada em uma ficha. O cliente atribui um valor (uma prioridade) à história baseando-se no valor de negócio global do recurso ou função. Os membros da equipe XP avaliam então cada história e atribuem um custo medido em semanas de desenvolvimento a ela. 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. É importante notar que podem ser escritas novas histórias a qualquer momento.

Para Ventura (2016, p. 1), “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”. Ainda segundo Ventura (2016, p. 3):

> Achar uma definição objetiva e exata para a palavra “requisito” é difícil. A palavra possui alguns significados (conforme os dicionários), mas chega a ser um pouco abstrata. Mas basicamente, quando falamos de requisito, estamos falando de necessidade, exigência, desejo, solicitação. 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, 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.

Continuando, Ventura (2016), defende e acredita, “que 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, Ventura (2016, p. 7) diz:

> É comum os profissionais de engenharia de software associarem a ideia de um Requisito Funcional a uma tela, uma rotina, que no fim serão as funcionalidades de fato de um sistema. Mas é necessário entender que uma funcionalidade não necessariamente realizará apenas um Requisito Funcional, podendo realizar vários requisitos funcionais – significa que em uma funcionalidade um ou mais requisitos funcionais podem ser atendidos, não necessariamente apenas um.

Sobre requisitos não funcionais Ventura (2016, p. 21) diz “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”. 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 (2009, p. 22):

> A fase de levantamento de requisitos deve identificar dois tipos de requisitos: os funcionais e os não-funcionais. Os requisitos funcionais correspondem ao que o cliente quer que o sistema realize, ou seja, as funcionalidades do software. Já os requisitos não-funcionais correspondem às restrições, condições, consistências, validações que devem ser levadas a efeito sobre os requisitos funcionais. Por exemplo, 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.
>
> Podem existir diversos tipos de requisitos não-funcionais, como de usabilidade, desempenho, confiabilidade, segurança ou interface. 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.

Ressaltando o levantamento e análise de requisitos, para Guedes (2009, p. 21):

> Uma das primeiras fases de um processo de desenvolvimento de software consiste no levantamento de requisitos. As outras etapas, sugeridas por muitos autores, são: Análise de Requisitos, Projeto, que se constitui na principal fase da modelagem, Codificação, Testes e Implantação. Dependendo do método/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. Se tomarmos como exemplo o Processo Unificado (*Unified Process*), um método de desenvolvimento de software, veremos que este se divide em quatro fases: 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. 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, volta-se à fase de Elaboração para tratar do ciclo seguinte, até todo o software ser finalizado.

Segundo Pádua (2000, p. 97):

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

Projetos de produtos mais complexos geralmente precisam de maior investimento em engenharia de requisitos que projetos de produtos mais simples. A Engenharia de Requisitos é também mais complexa no caso de produtos novos. 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. 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.

Uma boa engenharia de requisitos é um passo essencial para o desenvolvimento de um bom produto, em qualquer caso. Este capítulo descreve de forma detalhada as atividades do fluxo de Requisitos do Praxis, assim como algumas das técnicas mais importantes para a obtenção de requisitos de alta qualidade. 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.

Requisitos de alta qualidade são claros, completos, sem ambiguidade, implementáveis, consistentes e testáveis. Os requisitos que não apresentem estas qualidades são problemáticos, portanto, “eles devem ser revistos e renegociados com os clientes e usuários.

No Praxis, os requisitos estão contidos nos artefatos enumerados. 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. Os demais artefatos fazem parte das linhas de base. O enunciado detalhado dos requisitos estará contido na Especificação dos Requisitos do Software. O modelo dos casos de uso, parte da descrição dos requisitos funcionais, estará contido no Modelo de Análise do Software. 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 (2000, p. 99):

> Os requisitos de um produto podem alterar-se ao longo de seu desenvolvimento por diversos motivos:
> • descoberta de defeitos e inadequações nos requisitos originais;
> • falta de detalhes suficientes nos requisitos originais;
> • alterações incontornáveis no contexto do projeto (por exemplo, mudanças de legislação).
> Mesmo reconhecendo este fato, todo o esforço deve ser feito para que a especificação dos requisitos do software seja tão completa quanto possível. No caso de alterações serem indispensáveis, elas devem obedecer aos procedimentos de gestão de requisitos de software.
> Segundo o paradigma SW-CMM, uma organização considerada madura na gestão de requisitos de software deve atingir as seguintes metas:
> • Os requisitos de software são controlados para estabelecer uma base para as atividades gerenciais e de engenharia de software, dentro de um projeto.
> • Os planos, resultados, produtos e atividades de software são mantidos consistentes com os requisitos de software.

Segundo Pádua (2000, p. 100):

> 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 - Todo requisito presente na realmente é um requisito do produto a ser construído.
> • Precisa - Todo requisito presente possui apenas uma única interpretação, aceita tanto pelos desenvolvedores quanto pelos usuários chaves.
> • Completa - Reflete todas as decisões de especificação que foram tomadas.
> • Consistente - Não há conflitos entre nenhum dos subconjuntos de requisitos presentes.
> • Priorizada - Cada requisito é classificado de acordo com a sua importância, estabilidade e complexidade.
> • Verificável - Todos os seus requisitos são verificáveis.
> • Modificável - Sua estrutura e estilo permitem a mudança de qualquer requisito, de forma fácil, completa e consistente.

• Rastreável - Permite a fácil determinação dos antecedentes e consequências de todos os requisitos.

Pádua (2000, p. 101), comenta cada uma das características de qualidade conforme a seguir:

Correção

Uma Especificação dos Requisitos é correta se todo requisito presente nela realmente é um requisito do produto a ser construído. Não existe ferramenta que garanta a correção de uma Especificação dos Requisitos. 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. 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.

Precisão

Uma Especificação dos Requisitos é precisa se todo requisito presente possuir apenas uma única interpretação, aceita tanto pelos desenvolvedores quanto pelos usuários chaves. 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. Recomenda-se a inclusão no glossário da Especificação dos Requisitos de todos os termos contidos no documento que possam causar ambiguidades em sua interpretação. Os seguintes meios devem ser usados para garantir maior precisão da Especificação dos Requisitos:

• revisões técnicas;

• uso de notações e ferramentas de análise, orientadas a objetos no caso do Praxis.

Completeza

Uma Especificação dos Requisitos é completa se reflete todas as decisões de especificação que foram tomadas, não contendo cláusulas de pendências. Uma Especificação dos Requisitos completa deve:

• conter todos os requisitos significativos relativos à funcionalidade, desempenho, restrições de desenho, atributos e interfaces externas;

• definir as respostas do software para todas as entradas possíveis, válidas e inválidas, em todas as situações possíveis;

• conter um glossário de todos os termos técnicos e unidades de medida, assim como referências completas a todos os diagramas, figuras e tabelas.

Consistência

Uma Especificação dos Requisitos é consistente se não há conflitos entre nenhum dos subconjuntos de requisitos presentes. Existem três tipos comuns de conflitos entre requisitos:

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

• conflito 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;

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

Priorização

Uma Especificação dos Requisitos é priorizada se cada requisito é classificado de acordo com a respectiva importância e estabilidade. A estabilidade estima a probabilidade de que o requisito venha a ser alterado no decorrer do projeto, com base na experiência de projetos correlatos. A priorização classifica o requisito de acordo com um dos seguintes graus:

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

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

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

Verificabilidade

Uma Especificação dos Requisitos é verificável se todos os seus requisitos são verificáveis. 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. Em geral requisitos ambíguos não são verificáveis, assim como requisitos definidos em termos qualitativos, ou contrários a fatos técnicos e científicos.

Modificabilidade

Uma Especificação dos Requisitos é modificável se sua estrutura e estilo permitirem a mudança de qualquer requisito, de forma fácil, completa e consistente. A modificabilidade geralmente requer:

• organização coerente, com índices e referências cruzadas;

• ausência de redundância entre requisitos;

• definição separada de cada requisito.

Rastreabilidade

Uma Especificação dos Requisitos é rastreável se permite a fácil determinação dos antecedentes e consequências de todos os Requisitos. Dois tipos de rastreabilidade devem ser observados.

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

• Rastreabilidade para frente - deve ser possível localizar quais os resultados do desenvolvimento que serão afetados por cada requisito. 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.

2.5.1. Documentação

Segundo Pareto (2016, p. 21), 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, Helm, Johnson e Vlissides (2007, p. 18):

> Os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas. Os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização. 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. Em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.

Segundo Guedes (2009, p. 21):

> Por mais simples que seja, todo e qualquer sistema deve ser modelado antes de se iniciar sua implementação, entre outras coisas, porque os sistemas de informação frequentemente costumam ter a propriedade de “crescer”, isto é, aumentar em tamanho, complexidade e abrangência”. Muitos profissionais costumam afirmar que sistemas de informação são “vivos”, porque nunca estão completamente finalizados. Na verdade, o termo correto seria “dinâmicos”, pois normalmente os sistemas de informação estão em constante mudança”. Contudo, “tais mudanças são devidas a diversos fatores, como, por exemplo”:
> • Os clientes desejam constantemente modificações ou melhorias no sistema.
> • O mercado está sempre mudando, o que força a adoção de novas estratégias por parte das empresas e, consequentemente, de seus sistemas.
> • O 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.
> Assim, um sistema de informação precisa ter uma documentação extremamente detalhada, precisa e atualizada para que possa ser mantido com facilidade, rapidez e correção, sem produzir novos erros ao corrigir os antigos. 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.

Ainda segundo Guedes (2009, p. 20):

> Qual a real necessidade de se modelar um software? Muitos “profissionais” podem afirmar que conseguem determinar todas as necessidades de um sistema de informação de cabeça, e que sempre trabalharam assim. Mas a questão é muito mais ampla, envolvendo fatores extremamente complexos, como levantamento e análise de requisitos, prototipação, tamanho do projeto,

complexidade, prazos, custos, documentação, manutenção e reusabilidade, entre outros.

Contudo, Guedes (2009, p. 28) ressalta:

> 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, já que é muito comum que, depois de manter uma rotina ou função de um software, outras rotinas ou 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.
> 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, já que muitas vezes uma documentação desatualizada pode ser mais prejudicial à manutenção do sistema do que nenhuma documentação.

Para Guedes (2009, p. 29):

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:

• A média de manutenções que um sistema sofre normalmente dentro de um determinado período de tempo.
• Qual a média de custo de modelagem.
• Qual a média de custo de desenvolvimento.
• Qual a média de tempo despendido até a finalização do projeto.
• Quantos profissionais são necessários envolver normalmente em um projeto.

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. Além disso, a documentação pode ser muito útil em outra área: a Reusabilidade. Um dos grandes desejos e muitas vezes necessidades dos clientes é que o software esteja concluído o mais rápido possível. Uma das formas de agilizar é a reutilização de rotinas, funções e algoritmos previamente desenvolvidos em outros sistemas. Nesse caso, a documentação correta do sistema pode auxiliar a sanar questões como:

• Onde as rotinas se encontram?
• Para que foram utilizadas?
• Em que projetos estão documentadas?
• Elas são adequadas ao software atualmente em desenvolvimento?
• 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, Helm, Johnson e Vlissides (2007, p. 18):

> Os padrões de projeto tornam mais fácil reutilizar projetos e arquiteturas bem-sucedidas. Expressar técnicas testadas e aprovadas as torna mais acessíveis para os desenvolvedores de novos sistemas. Os padrões de projeto ajudam a escolher alternativas de projeto que tornam um sistema reutilizável e a evitar alternativas que comprometam a reutilização. 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. Em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.

2.5.2. Espeficação

Para Sommerville (2011, p. 39), “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 (2000, p. 39), “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 e Soares (2007, p. 20) “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, p. 18):

> Um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software. Essas atividades podem envolver o desenvolvimento de software a partir do zero em uma linguagem padrão de programação como Java ou C. No entanto, aplicações de negócios não são necessariamente desenvolvidas dessa forma. 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, p. 18):

> Existem muitos processos de software diferentes, mas todos devem incluir quatro atividades fundamentais para a engenharia de software. São ela:
> 1. Especificação de software. A funcionalidade do software e as restrições a seu funcionamento devem ser definidas.
> 2. Projeto e implementação de software. O software deve ser produzido para atender às especificações.
> 3. Validação de software. O software deve ser validado para garantir que atenda às demandas do cliente.
> 4. Evolução de software. O software deve evoluir para atender às necessidades de mudança dos clientes.

Continuando e ressaltando, segundo Sommerville (2011, p. 24), “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 e Soares (2007, p. 27), “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, p. 25):

> 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. Existem quatro atividades principais do processo de engenharia de requisitos:
>
> 1. 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. 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. 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.
> 2. 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, análise de tarefas,
> entre outras etapas. 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.
> 3. 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. Dois tipos de requisitos podem ser incluídos nesse documento. Requisitos do usuário são declarações abstratas dos requisitos do sistema para o

> cliente e usuário final do sistema; requisitos de sistema são uma descrição mais detalhada da funcionalidade a ser provida.
> 4. 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. Em seguida, o documento deve ser modificado para correção desses problemas.

Para Freitas (2015, p. 28):

> Ao final do processo de engenharia de software é obtida a especificação de um produto de software. Esta especificação deve garantir que as necessidades e expectativas do cliente e dos usuários do sistema sejam atendidas. 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. 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. A garantia de que a especificação atenda tanto cliente quanto usuários é realizada por meio de várias atividades já relacionadas.

Como já citado no capítulo sobre documentação, e pelo fato de se aplicar nesse presente capítulo, em se tratando de padrões de projetos, segundo Gamma, Helm, Johnson e Vlissides (2007):

> 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. Em suma, ajudam um projetista a obter mais rapidamente um projeto adequado.

Para Pádua (2000, p. 98):

A especificação dos requisitos do software é o documento oficial de descrição dos requisitos de um projeto de software. Ela pode se referir a um produto indivisível de software, ou a um conjunto de componentes de software, que formam um produto quando usados em conjunto (por exemplo, um módulo cliente e um módulo servidor).

As características que devem estar contidas na Especificação dos Requisitos do Software incluem":

• Funcionalidade: O que o software deverá fazer?

• Interfaces externas: Como o software interage com as pessoas, com o hardware do sistema, com outros sistemas e com outros produtos?

• Desempenho: Qual a velocidade de processamento, o tempo de resposta e outros parâmetros de desempenho requeridos pela natureza da aplicação?

• Outros atributos: Quais as considerações sobre portabilidade, manutenibilidade e confiabilidade que devem ser observadas?

• Restrições impostas pela aplicação: Existem padrões e outros limites a serem obedecidos, como linguagem de implementação, ambientes de operação, limites de recursos etc.?

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. O usuário chave é aquele que é indicado pelo cliente como pessoa capacitada a definir requisitos do produto, normalmente, os usuários chaves são escolhidos entre profissionais experientes das diversas áreas que usarão o produto. Estes usuários chaves devem ser devidamente informados e treinados sobre as técnicas e notações que serão utilizadas no fluxo de requisitos.

Geralmente, nem desenvolvedores nem clientes ou usuários são qualificados para escrever por si sós a especificação dos requisitos do software, porque:

• os clientes nem sempre entendem os processos de desenvolvimento de software em grau suficiente para produzir uma especificação de requisitos de implementação viável;

• os desenvolvedores nem sempre entendem a área de aplicação de forma suficiente para produzir uma especificação de requisitos satisfatória.

Os usuários chaves devem ser conscientizados do papel essencial que desempenham na especificação dos requisitos do software. 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), revisões técnicas e de

apresentação, avaliação das liberações, testes de aceitação e todos os procedimentos de implantação.

Para Pádua (2000, p. 99):

> O mais completo destes documentos é a especificação dos requisitos do sistema. Esta especificação definirá os requisitos aplicáveis ao sistema como um todo. Estes requisitos podem ser repassados aos componentes de software, ou realizados por outros componentes. Além disto, o desenho do sistema definirá as interfaces entre os componentes de software e os demais componentes. 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.
>
> Quando o software fizer parte de um sistema maior que está sendo especificado de forma concorrente, 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. Exemplos de equipes com as quais a equipe de especificação de software pode ter de interagir incluem os desenvolvedores de hardware, redes e bancos de dados e os especialistas da área de aplicação, além de pessoal de marketing e de áreas administrativas e financeiras.
>
> 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. 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. Todo requisito de sistema que tenha impacto no desenvolvimento de software deve ser aprovado pelo gerente do projeto de software.
>
> 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. Devem também estabelecer critérios de aprovação para cada componente do sistema que um grupo deva fornecer a outros grupos.

Porém, para Pádua (2000, p. 100):

> Normalmente, a especificação dos requisitos do software não deve incluir decisões de desenho e implementação, nem aspectos gerenciais de projeto. 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, determinados componentes ou determinadas plataformas de bancos de dados. Por isto, a especificação dos requisitos do software deverá satisfazer os seguintes critérios:

• Definir completa e corretamente todos os requisitos do produto do software. Requisitos podem existir em virtude da natureza do problema a ser resolvido, ou em virtude de outras características específicas do projeto.

• Não descrever qualquer detalhe de desenho ou de implementação. Estes devem ser descritos nos modelos e documentos produzidos pelos respectivos fluxos.

• Não descrever aspectos gerenciais do projeto, como custos e prazos. Estes devem ser especificadas em outros documentos, tais como o Plano de Desenvolvimento do Software ou o Plano da Qualidade do Software.

Normalmente, os seguintes itens são considerados como parte do desenho, e não devem fazer parte da especificação dos requisitos do software:

• partição do produto em módulos;

• alocação de funções aos módulos;

• fluxo de informação entre módulos;

• estruturas internas de dados".

Os requisitos a seguir são considerados requisitos gerenciais do projeto, e não devem ser incluídos na especificação dos requisitos do software:

• custo;

• cronograma de entregas;

• relatórios requeridos;

• métodos requeridos de desenvolvimento;

• procedimentos de controle da qualidade;

• critérios de verificação e validação".

2.6. Estratégica e organização do processo das rotinas

Para Pareto (2016, p. 32), "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, p. 177), 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. 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. 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, p. 19):

> Os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software. 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. Para alguns sistemas, como sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado, 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, p. 19):

> Embora não exista um processo ideal de software, há espaço, em muitas organizações, para melhorias no processo de software. Os processos podem incluir técnicas ultrapassadas ou não aproveitar as melhores práticas de engenharia de software da indústria. De fato, muitas empresas ainda não se aproveitam dos métodos da engenharia de software em seu desenvolvimento de software.
>
> Em organizações nas quais a diversidade de processos de software é reduzida, os processos de software podem ser melhorados pela padronização. Isso possibilita uma melhor comunicação, além de redução no período de treinamento, e torna mais econômico o apoio ao processo automatizado. 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, Helm, Johnson e Vlissides (2007, p. 324):

> Estudos de programadores especialistas em linguagens convencionais mostraram que o conhecimento e a experiência não são organizados simplesmente em torno da sintaxe, mas sim em torno de estruturas conceituais maiores, tais como algoritmos, estruturas de dados e termos, e planos para atingir um objetivo específico. 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, estrutura de dados e termos que aprenderam no passado.
>
> Os cientistas da computação nomeiam e catalogam algoritmos e estruturas de dados, porém, raramente nomeamos outros tipos de padrões. Os padrões de projeto fornecem um vocabulário comum para comunicar, documentar e explorar alternativas de projeto. 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. Os padrões de projeto elevam o nível no qual você projeta e discute o projeto com seus colegas.

Segundo Pádua (2000, p. 23):

> 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). Processos podem ser definidos com mais ou menos detalhes, como acontece com qualquer receita. Os passos de um processo podem ter ordenação apenas parcial, o que pode permitir paralelismo entre alguns passos.

Segundo Pádua (2000, p. 70):

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

Problemas de processo, são de responsabilidade dos gerentes dos projetos. Só através da melhoria de processos é possível evitar a repetição de erros e incorporar as lições dos projetos passados. O gerente de projeto que não tem como prioridade a melhoria dos processos está condenado a apagar incêndios eternamente. É papel do gerente de projeto propor a sua equipe metas desafiadoras, em termos de custos, prazos e qualidade dos resultados. Por outro lado, estas metas devem ser claramente viáveis, sob pena de não serem realmente levadas a sério. Uma vez propostas as metas de processo, elas devem ser cobradas, insistindo-se em criar uma cultura de excelência.

Naturalmente, erros e problemas acontecerão. O gerente de projeto deve então procurar as deficiências de processo que causaram os problemas. Gerentes que dão prioridade à caça de culpados estão, na realidade, convidando seus subordinados a esconder problemas. Gerentes que são os primeiros a quebrar as regras estão deixando bem claro que estas regras não são realmente para valer.

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 ser causados por quem teme a mudança, não a leva a sério, ou não a entende. Processos imaturos são baseados na improvisação individual. 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. 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.

A mudança eficaz requer conhecimento dos processos atuais. 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. Sem o entendimento destes, é difícil estabelecer prioridades de melhoria. 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.

> 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. Treinamento é fundamental: a melhoria dos processos envolve a absorção de conceitos que não são triviais, e que vão até contra a intuição de algumas pessoas. O programa de treinamento tem de ter planejamento, recursos, cronograma e obrigatoriedade. Algumas ferramentas têm de ser adquiridas e implantadas.
>
> O amadurecimento dos processos se faz passo a passo. Para ser viável, a melhoria deve ser feita em etapas planejadas. 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. Estes pontos podem corresponder a estágios intermediários de capacitação. A experiência da indústria de software deu origem a vários modelos de capacitação, que propõem áreas prioritárias para investimento em melhoria de processos.

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

2.6.1. Decisões estratégica

Segundo Pressman (2011, p. 38):

> Indivíduos, negócios e governos dependem, de forma crescente, de software para decisões estratégicas e táticas, assim como para controle e para operações cotidianas. 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, Pereira e Moritz (2010, p. 76):

> Os processos de tomadas de decisão são constantes no dia a dia organizacional e a todo o momento as pessoas estão sendo colocadas 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, p. 11):

> Um dos desafios da boa gestão na atualidade é, com certeza, o domínio do conhecimento de seus processos organizacionais, que tem grande importância como subsídio para diversas ações na organização, tais como o gerenciamento do desempenho, a tomada de decisão, o dimensionamento da força de trabalho, a desburocratização, a manutenção das rotinas, a melhoria dos serviços e produtos, a flexibilização organizacional, entre outros. 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”.

2.6.2. Opções de estratégicas

Para Sommerville (2011, p. 177), existem quatro opções estratégicas:

> 1. Descartar completamente o sistema. Essa opção deve ser escolhida quando o sistema não está mais contribuindo efetivamente para os processos dos negócios. Isso geralmente ocorre quando os processos de negócios se alteram desde que o sistema foi instalado e já não são dependentes do sistema legado.
> 2. Deixar o sistema inalterado e continuar com a manutenção regular. 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.
> 3. Reestruturar o sistema para melhorar sua manutenibilidade. 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. Esse processo pode incluir o desenvolvimento de novos componentes de interface, para que o sistema original possa trabalhar com outros sistemas mais novos.
> 4. Substituir a totalidade ou parte do sistema por um novo sistema. Essa opção deve ser escolhida quando fatores como hardwares novos significam que o

> sistema antigo não pode continuar em operação ou quando sistemas de prateleira podem permitir o desenvolvimento do novo sistema a um custo razoável. Em muitos casos, uma estratégia de substituição evolutiva pode ser adotada, na qual, sempre que possível, os componentes principais do sistema são substituídos por sistemas de prateleira com outros componentes reusados.

2.6.3. Organização e centralização dos processos

Sobre o modelo de processos, Pressman (2011, p. 78), explica que:

> Os modelos de processo prescritivos são aplicados há anos, num esforço para organizar e estruturar o desenvolvimento de software. Cada um desses modelos sugere um fluxo de processos ligeiramente diferente, mas todos realizam o mesmo conjunto de atividades metodológicas genéricas: comunicação, planejamento, modelagem, construção e emprego.

Para Marks (2008, p. 20), “a centralização, nas organizações, tem a ver com a distribuição”. Ainda de acordo com Marks (2008, p. 29):

> Seja a empresa de porte pequeno, seja de porte médio, seja de porte grande, para funcionar bem ela necessita de uma estrutura organizacional adequada. 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, p. 48), que diz o seguinte:

> Departamentalizar é dividir a empresa em órgãos ou unidades organizacionais. Em cada órgão são realizadas atividades afins. Por exemplo, no departamento financeiro são realizadas todas as atividades relacionadas com as finanças da empresa, assim como no departamento de vendas são tratadas as questões relacionadas às vendas. 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, p. 48), “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, p. 48):

> As atividades intermediárias geralmente realizam o processamento, ou seja, algum tipo de trabalho de transformação ou modificação. 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. Nas atividades iniciais ocorrem as entradas do processo e nas finais ocorrem as saídas. As entradas correspondem a tudo o que o processo recebe e as saídas, ao que ele fornece. O grande processo geral de uma empresa pode compor-se de um conjunto de processos menores.

Para Armelin, Pelegrini e Colucci (2016, p. 70):

> Um sistema empresarial constitui uma estrutura centralizada para uma organização e 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. Os sistemas empresariais empregam um banco de dados operacional e de planejamento fundamentais que podem ser compartilhados por todos. Isso elimina os problemas de informações inconsistentes causados por múltiplos sistemas de processamento de transação, que atuam somente em uma função do negócio ou em função de um departamento de uma organização.
>
> Quanto aos benefícios de um Sistema Empresarial podemos elencar:
>
> • Auxílio na tomada de decisão empresarial.
>
> • Aumento do valor agregado aos produtos, bens e serviços prestados pela empresa.
>
> • Criação ou desenvolvimento de uma vantagem competitiva.
>
> • Desenvolvimento e/ou fabricação de produtos com maior qualidade.
>
> • Maior oportunidade de negócios, uma vez que se conhece o negócio da empresa.
>
> • Maior rentabilidade.
>
> • Maior precisão de informações, com isso, redução de erros.
>
> • Diminuição de desperdícios e com isso, menores custos.

Para Sommerville (2011, p. 8), existem fundamentos de engenharia de software que se aplicam a todos os tipos de sistemas de software:

> 1. Eles devem ser desenvolvidos em um processo gerenciado e compreendido. A organização que desenvolve o software deve planejar o processo de desenvolvimento e ter ideias claras do que será produzido e quando estará finalizado. É claro que processos diferentes são usados para tipos de software diferentes.
> 2. Confiança e desempenho são importantes para todos os tipos de sistema. O software deve se comportar conforme o esperado, sem falhas, e deve estar disponível para uso quando requerido. 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. É importante entender e gerenciar a especificação e os requisitos de software (o que o software deve fazer). Você deve 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. Você deve fazer o melhor uso possível dos recursos existentes. Isso significa que, quando apropriado, você deve reusar o software já desenvolvido, em vez de escrever um novo.

Segundo Sommerville (2011, p. 19):

> Os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software. 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. Para alguns sistemas, como sistemas críticos, é necessário um processo de desenvolvimento muito bem estruturado; para sistemas de negócios, com requisitos que se alteram rapidamente, provavelmente será mais eficaz um processo menos formal e mais flexível.
>
> Os processos de software, às vezes, são categorizados como dirigidos a planos ou processos ágeis. Processos dirigidos a planos são aqueles em que todas as atividades são planejadas com antecedência, e o progresso é avaliado por

> comparação com o planejamento inicial. Em processos ágeis, o planejamento é gradativo, e é mais fácil alterar o processo de maneira a refletir as necessidades de mudança dos clientes.
>
> Embora não exista um processo 'ideal' de software, há espaço, em muitas organizações, para melhorias no processo de software. Os processos podem incluir técnicas ultrapassadas ou não aproveitar as melhores práticas de engenharia de software da indústria. De fato, muitas empresas ainda não se aproveitam dos métodos da engenharia de software em seu desenvolvimento de software.
>
> Em organizações nas quais a diversidade de processos de software é reduzida, os processos de software podem ser melhorados pela padronização. Isso possibilita uma melhor comunicação, além de redução no período de treinamento, e torna mais econômico o apoio ao processo automatizado. 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 (2000, p. 80) cita os 5 níveis de organização, que são:

> A organização nível 1 representa o estágio inicial dos produtores de software. Ela utiliza processos informais e métodos ad hoc, às vezes descritos como caóticos. Muitas destas organizações são bem-sucedidas, já que o mercado de software é ainda extremamente tolerante em relação à má qualidade dos produtos. 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. É comum a dependência em relação a heróis técnicos e gerenciais. 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. 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. Engenharia de requisitos e desenho são fracos ou inexistentes; mudanças de requisitos e de outros artefatos ocorrem sem controle. A instalação e manutenção costumam ser deficientes, sendo encaradas como atividades de pouca importância. Os gerentes destas organizações geralmente não entendem os verdadeiros problemas, por falta de processos que lhes deem visibilidade real em relação ao progresso dos projetos.

São comuns os casos de gerentes com formação exclusivamente administrativa, que não entendem os problemas técnicos dos projetos. Existem também aqueles que chegaram a gerentes como promoção da carreira técnica, e não têm a mínima formação em práticas gerenciais. 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. Na melhor das hipóteses, os processos são seguidos quando os projetos estão em fase tranquila; em crise, abandonam-se os métodos, e reverte-se à codificação desenfreada.

A tônica da organização nível 2 é ser capaz de cumprir compromissos. 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. Isto requer o domínio das seguintes áreas chaves:

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

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

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

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

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

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

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:

• mudanças de ferramentas e métodos, trazida pela evolução de tecnologia;

• mudanças de tipos de produto, causadas por variações dos mercados;

• mudanças de estrutura organizacional, causadas por diversos fatores da dinâmica das organizações.

O nível 3 conduz da gestão de projetos à engenharia de produtos. Este nível de organização não repete simplesmente os sucessos de projetos anteriores, mas estabelece uma infraestrutura de processos que permite a adaptação a vários tipos de mudanças. Este nível de organização requer o domínio das seguintes áreas chaves:

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

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

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

• 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;

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

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

• coordenação de revisões técnicas a nível da organização.

A organização nível 3 sabe manter-se dentro do processo, mesmo durante as crises. As ferramentas passam a ser aplicadas de forma sistemática, padronizada e coerente com os processos. 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. Existe uma base de dados de processos, povoada com os dados recolhidos dos projetos; esta base é usada para gestão dos projetos, mas não é ainda aplicada, de forma sistemática e a nível da organização, para atingir metas quantitativas de desempenho de processo e de qualidade de produto.

Na organização nível 4, o domínio dos processos de software evolui para uma forma quantitativa. Isto não quer dizer que apenas organizações deste nível devam coletar métricas de processo. 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. A organização nível 3 constrói e mantém uma base de dados de processos. Coleta de dados é uma atividade cara: é necessário definir com precisão e antecipação os dados que vão

ser coletados. Estes dados têm de ser criticados, consistidos e normalizados para terem alta qualidade. 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. Além disto, sabe intervir nos processos para atingir metas de qualidade dos produtos. Este nível tem apenas duas áreas chaves:

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

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

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:

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

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

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

2.6.4. Visão e adaptação do sistema a nível organizacional

Para Sommerville (2011, p. 39):

> 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, 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, p. 454):

> O gerenciamento de qualidade está preocupado com o estabelecimento de um framework de processos organizacionais e padrões que levem a softwares de alta qualidade. Isso significa que a equipe de gerenciamento de qualidade deve assumir a responsabilidade de definir os processos de desenvolvimento do software 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, p. 298):

> O reuso de software é mais eficaz quando está previsto como parte de um programa de reuso de toda a organização. 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, Pelegrini e 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 (2000), ressalta a maturidade das organizações e seus sintomas:

> A produção industrial de software é quase sempre uma atividade coletiva. Alguns produtos são construídos inicialmente por indivíduos ou pequenas equipes. Na medida em que se tornam sucesso de mercado, passam a evoluir. A partir daí um número cada vez maior de pessoas passa a cuidar da manutenção e evolução dele. Por isso, quase todas as atividades de Engenharia de Software são empreendidas por organizações.
>
> 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 Pádua (2000, p. 57):

Infelizmente para os profissionais, muitas organizações que produzem software são imaturas. Isto ocorre tanto com organizações que produzem software como atividade fim, como com organizações para as quais o software é meio de apoio aos processos de negócio. Alguns sintomas identificam claramente as organizações imaturas:

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

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

• As 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. Os usuários das ferramentas não recebem treinamento e orientação em grau satisfatório. Ferramentas são escolhidas de forma política, sem considerar avaliações técnicas e necessidades de padronização.

• Os procedimentos e padrões, quando existem, são definidos e seguidos de forma burocrática. Muitas vezes, o processo oficial, que existe nos documentos escritos, é rígido demais. Por outro lado, o processo real praticado é, com frequência, muito diferente do processo oficial, e muito mais relaxado que este. Os gerentes são os primeiros a não levar os processos a sério.

2.6.5. Refatoração

Para Fowler (2000, p. 9),

> 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.

> 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), “a noção de refatoração, ou seja, melhoria da estrutura e organização de um programa, também é um importante mecanismo que suporta mudanças”. Será que a não alteração do código fonte do software nesse processo de refatoração, está organizando e de certa maneira essa organização contribui para a centralização das rotinas da organização? Será que esse processo de melhorar na refatoração de certa maneira não estar agilizando e mantendo os processos do software, sistema legado da organização?

Gamma, Helm, Johnson e Vlissides (2007, p. 325), ressaltam que:

> Um dos problemas no desenvolvimento de software reutilizável é que, frequentemente, 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.

Ainda segundo Gamma, Helm, Johnson e Vlissides (2007, p. 326):

> 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 frequentemente 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, frequentemente 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.

Para Fowler (2000, 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”.

Ainda de acordo com Fowler (2000, p. 48):

> A refatoração torna o software mais fácil de entender a programação. É, sob muitos aspectos, uma conversa com um computador. Você escreve um código que diz ao computador o que fazer e responde fazendo exatamente o que você diz. Com o tempo você fecha a lacuna entre o que você quer fazer e o que você diz para fazer. A programação neste modo tem tudo a ver dizendo exatamente o que você quer. Mas há outro usuário do seu código-fonte. Alguém tentará ler seu código dentro de alguns meses para fazer algumas alterações. Esquecemos facilmente que o usuário é realmente o mais importante. Quem se importa se o computador leva alguns mais ciclos para compilar algo? Não importa se leva um programador por semana para fazer uma mudança que levaria apenas uma hora se ela tivesse entendido seu código.
>
> O problema é que, quando você está tentando fazer o programa funcionar, você não está pensando nisso futuro desenvolvedor. É preciso uma mudança de ritmo para fazer alterações que facilitam o entendimento do código. A refatoração ajuda você a tornar seu código mais legível. Ao refatorar você tem código que funciona, mas não está idealmente estruturado. Um pouco de tempo gasto na refatoração

> pode tornar o código comunicar melhor o seu propósito. Programar neste modo tem a ver com dizer exatamente o que você significa.
>
> Eu não estou necessariamente sendo altruísta sobre isso. Muitas vezes, esse futuro desenvolvedor sou eu. Aqui refatoração é particularmente importante. Eu sou um programador muito preguiçoso. Uma das minhas formas de preguiça é que nunca lembre-se de coisas sobre o código que escrevo. Na verdade, eu deliberadamente tento não me lembrar de nada que possa olhe para cima, porque temo que meu cérebro fique cheio. Faço questão de colocar tudo o que devo lembre-se do código para que eu não precise me lembrar dele. Dessa forma, eu estou menos preocupado com matando as células do meu cérebro. Esse entendimento também funciona de outra maneira. Uso refatoração para me ajudar a entender coisas desconhecidas, código. Quando olho para código desconhecido, tenho que tentar entender o que ele faz. Eu olho para um par de linhas e diga para mim mesmo: ah, sim, é isso que esse pedaço de código está fazendo. Com a refatoração, eu não paro em a nota mental. Na verdade, mudo o código para refletir melhor minha compreensão e depois testo reexecutando o código para verificar se ele ainda funciona. No início, refatoro assim com pequenos detalhes. À medida que o código fica mais claro, percebo que posso ver coisas sobre o design que eu não podia ver antes. Se eu não tivesse mudado o código, provavelmente nunca mudaria essas coisas, porque eu não sou inteligente o suficiente para visualizar tudo isso na minha cabeça. Descreve essas refatorações iniciais como limpando a sujeira de uma janela para que você possa ver além. Quando estou estudando código, acho que a refatoração me leva a níveis mais altos de compreensão que caso contrário, eu sentiria falta.

Segundo Gamma, Helm, Johnson e Vlissides (2007, p. 325):

> Um dos problemas no desenvolvimento de software reutilizável é que, frequentemente, 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.

2.7. Algumas considerações sobre o capítulo

O referencial citou até aqui, as ideias principais de autores que atuam diretamente na área de engenharia de software, que visa levantar opiniões de pesquisa sobre o tema, a engenharia de software, desenvolvendo rotinas em PL/SQL, estrategicamente para

agilizar e manter processos. Conceituou e mostrou de forma sucinta questões relevantes pertinentes ao tema, mostrando bases ao longo do referencial a ligação direta com o tema proposto. Foi realizado ao longo do referencial pergunta para ser pensar sobre futuras pesquisas sobre o tema abordado.

3. PROCEDIMENTOS METODOLÓGICOS

De caráter formal exploratório, este trabalho, que disserta sobre o presente tema, engenharia de software, desenvolvendo rotinas em PL/SQL, estrategicamente para agilizar e manter processos, 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 agilizar e manter processos em software, sistemas legados de organizações do brasil. 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 Lakatos e Marconi (2003), “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 Lakatos e Marconi (2003, p. 201):

Questionário é um instrumento de coleta de dados, constituído por uma série ordenada de perguntas, que devem ser respondidas por escrito e sem a presença do entrevistador. 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. Análise de conteúdo permite a descrição sistemática, objetiva e quantitativa do conteúdo da comunicação. Análise de conteúdo, extrair generalizações com o propósito de produzir categorias conceituais que possam vir a ser operacionalizadas em um estudo subsequente.

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 PL/SQL 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. APRESENTAÇÃO E 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 5 perguntas, uma das 10 pessoas que responderam, tinha formação incompleta na área de tecnologia, 9 estava ou já tinha concluído sua formação na área, ressaltando que 1 das 10 pessoas optou por não responder uma das 5 perguntas do questionário. Isso indica que para um grupo de pessoas a capacitação do profissional em engenharia de software é uma necessidade para as empresas do Brasil. E esse entendimento é bom, para criar e melhorar rotinas do sistema legado da organização, de forma organizada e com qualidade.

4.2. Experiências e 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 futuros já mencionado por Sommerville (2011), “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 fortalecimento do legado da engenharia de software.

4.3. Análise dos resultados à luz documental

Para Lakatos e Marconi (2003, 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”.

> A técnica da observação não estruturada ou assistemática, também denominada espontânea, informal, ordinária, simples, livre, ocasional e acidental, consiste em recolher e registrar os fatos da realidade sem que o pesquisador utilize meios técnicos especiais ou precise fazer perguntas diretas. É mais empregada em estudos exploratórios e não tem planejamento e controle previamente elaborados (Lakatos e Marconi, 2003, p. 192)

Nota-se Lakatos e Marconi (2003, 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 Marques (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 Lakatos e Marconi (2003) “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 Lakatos e Marconi (2003), “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 ideias 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 ideias 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 (Lakatos e Marconi, 2003, 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 cinco perguntas básicas, feito com uma equipe de dez pessoas, nas quais eram ou estavam se formando na área de tecnologia. 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 ir em buscar da pesquisa e investigação. Motivaram e continuam me motivando. Pois, foi levantado a importância de solucionar o problema que é a qualidade do software empresarial. Essas respostas foram cruciais para levantar a relevância do tema pesquisado e escolha da fonte de pesquisa e coleta de dados. Entretanto, Segundo Lakatos e Marconi (2003), 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.

i) Para Sommerville (2011, p. 2), “o progresso da engenharia de software foi excepcional durante a minha carreira. Nossas sociedades não poderiam funcionar sem os grandes e profissionais sistemas de software”. Ressalta ainda que, “a engenharia de software é, portanto, uma tecnologia de importância crítica para o futuro da humanidade. Devemos

continuar a educar engenheiros de software e a desenvolver a disciplina para podermos criar sistemas de software mais complexos”

Pergunta: As organizações do Brasil que possuem sistemas legados necessitam hoje de profissionais capacitados na área de engenharia de software?

ii) Para Sommerville (2011, p. 4):

> 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.

Pergunta: É importante organizar e centralizar o código fonte do software da organização?

iii) Para Sommerville (2011, p. 31), “padrões de qualidade organizacional geralmente são relaxados para o desenvolvimento do protótipo”.

Pergunta: É importante saber padrões de engenharia de software para atuar no processo de melhoria de um software empresarial?

iv) Para Sommerville (2011, p. 5), “para acomodar os diferentes níveis de habilidade, alguns programadores não fazem refatoração em partes do sistema que não desenvolveram”.

Pergunta: Através do estudo se adquire o conhecimento e através da pratica adquire habilidades para ser um bom profissional?

v) **Pergunta**: Você é ou está se formando na área de tecnologia?.

4.5. Análise dos dados

Para Lakatos e Marcodi (2003, p. 167), “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 pelas as cores: vermelha para não e azul para sim. A análise dos dados coletados foram conforme as respostas, onde de dez participantes, foram obtidos de cinco perguntas 100% de aproveitamento. Onde a primeira pergunta teve 90% de resultado (sim) e 10% de (não), a segunda pergunta teve 88,9% de resultado (sim) e 11,1% de (não), a terceira pergunta teve 100% de resultado (sim), a quarta pergunta teve 100% de resultado (sim), a quinta e última pergunta teve 90% de resultado (sim) e 10% de (não). Os dados foram analisados com a análise de conteúdo, segundo Chizzotti (2006, 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 Badin (2011) é "um conjunto de técnicas de análise de comunicação", "que contem informação sobre o comportamento humano atestado por uma fonte documental".

O link para visualização do questionário encontra-se disponível no endereço seguinte https://forms.gle/uL5Lf59JPRGEDzVG7.

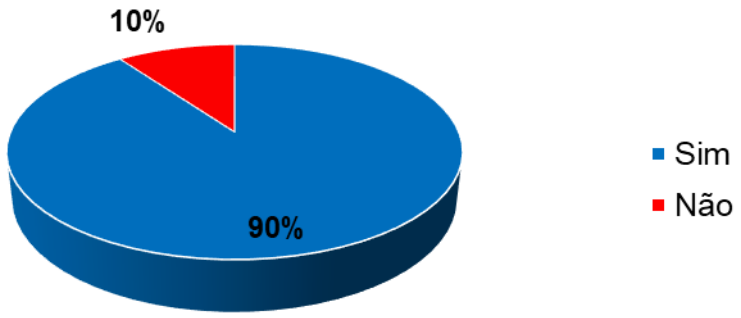

Figura 1. Pergunta 1: As organizações do Brasil que possuem sistemas legados necessitam hoje de profissionais capacitados na área de engenharia de software?.

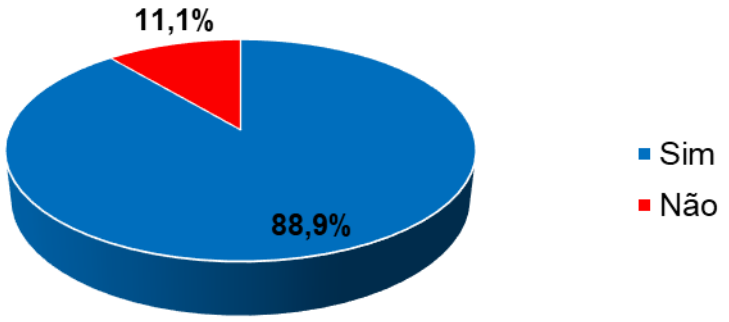

Figura 2. É importante organizar e centralizar o código fonte do software da organização?

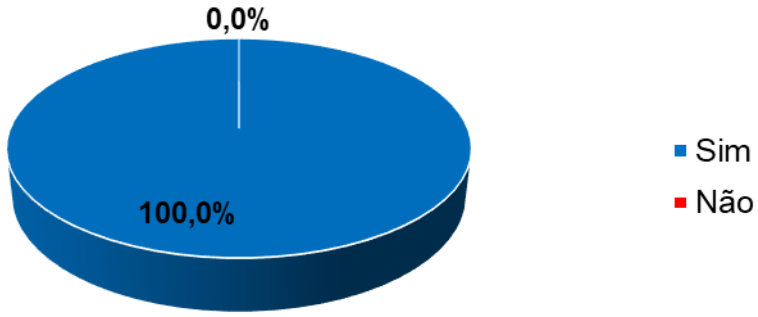

Figura 3. É importante saber padrões de engenharia de software para atuar no processo de melhoria de um software empresarial?

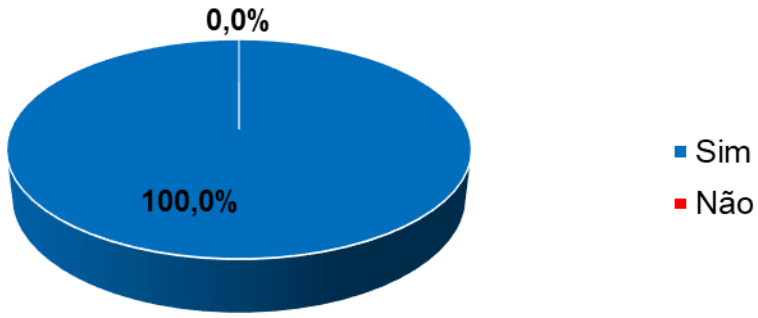

Figura 4. Através do estudo se adquire o conhecimento e através da pratica adquire habilidades para ser um bom profissional?

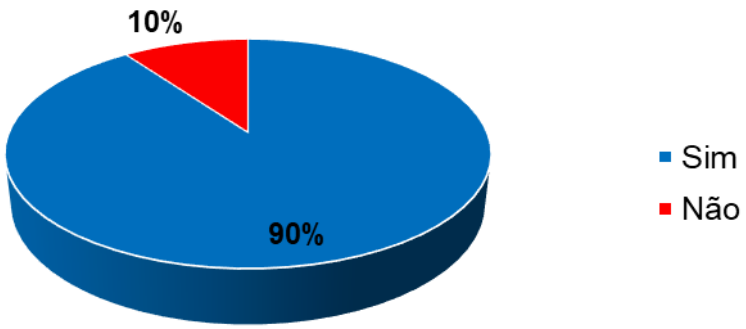

Figura 5. Você é ou está se formando na área de tecnologia?

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 uma rotina com base na linguagem PL/SQL, 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 PL/SQL. Todas essas estruturas são chamadas de máquina virtual.

5. CONSIDERAÇÕES FINAIS

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

5.1. Conclusões

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 de organizações do Brasil, se isso manteria seus processos com qualidade e agilidade, 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 ideias 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: Engenharia de Software: Desenvolvendo Rotinas em PL/SQL, Centralizadas Estrategicamente Para Agilizar e Manter Processos e por fim originou-se o trabalho aqui presente.

Diante disso a pesquisa teve como objetivo geral, analisar o desenvolvimento de rotinas, centralizadas estrategicamente para agilizar e manter processos. Com o objetivo de responder nos padrões da engenharia de software como realizar essa analise baseada nas ideias 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 conceito de desenvolvimento na engenharia.

O objetivo especifico inicial era descrever as características de desenvolvimento de rotinas de software, ele foi atendido por apresentar no tópico revisão da literatura em qualidade no desenvolvimento de rotina as suas características.

O segundo objetivo especifico era identificar as maiores dificuldades de manutenção das rotinas do sistema legado, ele foi atendido por apresentar no tópico tema/área de pesquisa e contextualização suas dificuldades.

O terceiro objetivo especifico era apresentar pontos relevantes de pesquisas para processos de engenharia de software, ele foi atendido por apresentar no tópico revisão da literatura em estratégica e organização do processo das rotinas seus pontos relevantes de pesquisas.

O quarto objetivo especifico era responder como organizar rotinas com base na pesquisa de engenharia de software, ele foi atendido por apresentar no tópico revisão da literatura em estratégica e organização do processo das rotinas os conceitos de organização de rotinas na engenharia de software.

O quinto e último objetivo especifico era averiguar até que ponto o desenvolvimento é essencial para o software organizacional, ele foi atendido por apresentar no tópico revisão da literatura em a evolução dos sistemas legados a importância do desenvolvimento organizacional.

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 rafaturamento abordando dentro da engenharia de software. Porque essa ideia 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 desconhecendo a importância dos padrões de qualidade no processo de melhoria das rotinas do sistema legado organizacional, então fez-se o teste da hipótese no capitulo procedimentos metodológicos onde a hipótese foi confirmada por isso.

Em resposta ao problema em certa parte encontramos um norte a ser seguido, porém não sabemos ao certo se foi respondida já que a ideia é fornecer um material para estudos. E pessoas da área de tecnologia precisam saber da existência desse material 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 centralizar em todo o trabalho as ideias 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, 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 dez pessoas contribuiu com a pesquisa respondendo-o.

5.2. 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 a limitação de tempo e recursos, só foi possível realizar com a quantidade limite proposta.

5.3. Recomendações

Recomendo pesquisas futuras sobre o tema proposto, sugiro que alimentem e aprofundem os métodos aqui abordados, pesquisando sobre performance das rotinas com uma grande quantidade de dados armazenado, hoje por estar dentro de uma organização de grande porte, vejo que existem uma dificuldade grande em seu sistema legado no quesito performance e retorno das informações, os grandes volumes de dados representam um desafio real para os sistemas legados das organizações.

Sugiro para futuras pesquisas sobre o tema também, apresentar conceitos de software gratuitos para pequena e média empresas, na grande maioria é possível trabalhar com bons resultados com software com versões gratuitas. Em muitos casos empresas podem ter software não tão robustos e com qualidade, sem a necessidade de pagar tão caro por um sistema e depois conforme o crescimento da empresa migrar para um software com capacidade de armazenamento de dados maior.

6. REFERÊNCIAS BIBLIOGRÁFICAS

Armelin, D. A., Silva, S. C. P., e Colucci, C. (2016). *Sistemas de informação gerencial*. Londrina: Kls.

Bardin, L. (2011). *Análise de conteúdo*. São Paulo: Edições 70.

Chizzotti, A. (2006). *Pesquisa em ciências humanas e sociais* (2ª ed.). São Paulo: Cortez.

Córdova, P. R. (2020). *A aprendizagem Baseada em Problemas (PBL) e a engenharia de software*. São Paulo: Paco Editorial.

Engholm, H. J. (2010). *Engenharia de software na prática*. São Paulo: Novatec.

Fernandes, L. (2002). *Oracle 9i Para desenvolvedores*. Rio de Janeiro: Axcel Books.

Ferreira, A. R. (2016). *Análise e melhoria de processos*. Brasília: Enap.

Fowler, M. (2000). *Refatoração: Aperfeiçoamento e Projeto*. São Paulo: Artmed Editora.

Freitas, R. R. (2015). *Análise e projeto de software*. Cuiabá: Rede e-Tec.

Fulgencio, P. C. (2007). *Glossário - Vade Mecum*. Rio de Janeiro: Mauad.

Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (2007). *Padrões de projeto, soluções reutilizáveis de software orientado a objetos*. Porto Alegre: Bookman.

Gil, A. C. (2017). *Como elaborar projetos de pesquisa* (6ª ed.). São Paulo: Atlas.

Gonçalves, E. (2015). *PL/SQL: Domine a linguagem do banco de dados Oracle*. São Paulo: Casa do Código.

Guedes, G. T. A. (2009). *UML 2: uma abordagem prática*. São Paulo: Novatec.

Koscianski, A. e Soares, M. S. (2007). *Qualidade de software: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software* (2ª ed.). São Paulo: Novatec.

Lobo, E. J. R. (2009). *Guia prático de engenharia de software*. São Paulo: Digerati Books.

Marconi, M. A. e Lakatos, E. M. (2003). *Fundamentos de metodologia científica*. São Paulo: Atlas.

Marks, S. R. (2008). *Estrutura e processos organizacionais*. Ijuí: Editora Unijuí.

Pádua, W. P. F. (2000). *Engenharia de software: fundamentos, métodos e padrões*. Rio de Janeiro: Editora Ltc.

Pareto, E. (2016). *Cenários de TI* (1ª ed.). Rio de Janeiro: Seses.

Pressman, R. S. (2011). *Engenharia de software: uma abordagem profissional* (7ª ed.). São Paulo: Amgh Editora Ltda.

Pressman, R. S. e Maxim, B. R. (2016). *Engenharia de software: uma abordagem profissional* (8ª ed.). São Paulo: Amgh Editora Ltda.

Préve, A. D., Moritz, G. O., e Pereira, M. F. (2010). *Organização, processos e tomada de decisão*. Florianópolis: Cad UFSC.

Price, J. (2009). *Database 11g SQL Domine SQL e PL/SQL no banco de dados Oracle*. Porto Alegre: Bookman.

Silva, A. M. (2015). *Metodologia da pesquisa* (2ª ed.). Ceará: Eduece.

Sommerville, I. (2011). *Engenharia de software* (9ª ed.). São Paulo: Pearson Education.

Ventura, P. (2016). *Ebook Requisitos de Software*. Belo Horizonte: Indtech.

ANEXOS

ANEXO 1: Declaração juramentada

Data: 24 de março de 2020

Em Fortaleza, Brasil, eu, Miguel Marcelo Nascimento Franco, identificado com o documento de matricula BRMDEISW2759383, estudante do programa Máster en Dirección Estratégica en Ingeniería de Software, titulado pela Universidad Europea del Atlántico declaro:

Que a pesquisa Engenharia de software: Desenvolvendo rotinas em pl/sql, centralizada estrategicamente para agilizar e manter processos, cujo objetivo principal é desenvolver e apresentar idéias e pensamentos de autores de grandes obras da engenharia de software para melhor organizar o código fonte de programação para centralizar a regra de negócio da organização, e da qual sou o pesquisador principal, não necessita da aprovação do Comitê de Ética enquanto não envolver a participação de seres humanos.

Miguel Marcelo Nascimento Franco

ANEXO 2: Questionário

Solicitamos a sua colaboração e sinceridade na participação desse questionário que se enquadra numa investigação no âmbito do Mestrado em Direção Estratégica em Engenharia de Software, da Universidad Europea del Atlántico. 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 anónima 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) Para Sommerville (2011, p. 2), “o progresso da engenharia de software foi excepcional durante a minha carreira. Nossas sociedades não poderiam funcionar sem os grandes e profissionais sistemas de software”. Ressalta ainda que, “a engenharia de software é, portanto, uma tecnologia de importância crítica para o futuro da humanidade.Devemos continuar a educar engenheiros de software e a desenvolver a disciplina para podermos criar sistemas de software mais complexos”.

Pergunta: As organizações do Brasil que possuem sistemas legados necessitam hoje de profissionais capacitados na área de engenharia de software?

2) Para Sommerville (2011, p. 4), “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”.

Pergunta: É importante organizar e centralizar o código fonte do software da organização?

3) Para Sommerville (2011, p. 31), “padrões de qualidade organizacional geralmente são relaxados para o desenvolvimento do protótipo”.

Pergunta: É importante saber padrões de engenharia de software para atuar no processo de melhoria de um software empresarial?

4) Para Sommerville (2011, p. 5), “para acomodar os diferentes níveis de habilidade, alguns programadores não fazem refatoração em partes do sistema que não desenvolveram”.

Pergunta: Através do estudo se adquire o conhecimento e através da pratica adquire habilidades para ser um bom profissional?

5) Pergunta: Você é ou estar se formando na área de tecnologia?

Obrigado