INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JOSIAS ANDRIEL GRABOSKI

ANÁLISE DO PROCESSO DE DESENVOLVIMENTO E SUPORTE A SERVIÇOS DE SOFTWARE NAS EMPRESAS DE PASSO FUNDO

Carmen Vera Scorsatto

PASSO FUNDO, 2013

JOSIAS ANDRIEL GRABOSKI

ANÁLISE DO PROCESSO DE DESENVOLVIMENTO E SUPORTE A SERVIÇOS DE SOFTWARE NAS EMPRESAS DE PASSO FUNDO

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio-Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador (a): Carmen Vera Scorsatto

PASSO FUNDO, 2013

JOSIAS ANDRIEL GRABOSKI

ANÁLISE DO PROCESSO DE DESENVOLVIMENTO E SUPORTE A SERVIÇOS DE SOFTWARE NAS EMPRESAS DE PASSO FUNDO

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Orientador Prof. Carmen Vera Scorsatto

_____

Convidado Fernando Abrahão Afonso

_____

Convidado André Fernando Rollwagen

_____

Prof. Me. Evandro Miguel Kuszera

Coordenador do Curso

PASSO FUNDO, 2013

DEDICATÓRIA

Aos meus pais e aos meus irmãos,
assim bem como aos meus amigos que sempre
estão ao meu lado passando
confiança e afastando meus receios.
Dedico a eles pela compreensão e o estímulo
em todos os momentos.

AGRADECIMENTOS

Quero agradecer em primeiro momento a Deus pelo fato de sempre ter me possibilitado condições de ir à busca de meus objetivos e por todos os desafios que ele me impõe, pois através deles é que eu cresço profissionalmente e como ser humano.

Gostaria de agradecer a minha orientadora Carmen Vera Scorsatto por todo apoio, disponibilidade e por ser sempre atenciosa quanto às atividades de meu projeto, a quem serei grato por toda vida por me auxiliar neste momento tão importante de minha carreira acadêmica.

Agradeço também ao Sr. Fernando Winckler Simor por toda contribuição prestada, tanto com suas experiências profissionais, quanto com sua concepção de mercado sobre o tema proposto, sua colaboração foi determinante para a conclusão do projeto.

Deixo meu agradecimento a todas as empresas que se disponibilizaram a responder o questionário proposto e contribuíram com informações que, geralmente, são difíceis de ser abertas pelas organizações. Sem elas seria impossível construir este trabalho, desejo a todos muito sucesso.

Por fim, agradeço a todos os que acreditaram em mais este objetivo de minha vida e que estão sempre presentes em todos os momentos, o bem mais precioso que possuo meus pais, irmãos e amigos. A todos meu sincero muito obrigado e que Deus vos ilumine sempre.

EPÍGRAFE

“Seus clientes menos satisfeitos são
sua maior fonte de aprendizado.”
Bill Gates

RESUMO

O trabalho Análise do Processo de Desenvolvimento e Suporte a Serviços de Software nas empresas de desenvolvimento de Passo Fundo, apresenta-se inicialmente com um estudo de campo com o intuito de investigar como é gerido o processo de software e o processo de manutenção nas empresas da cidade. Apresentando, posteriormente, uma descrição de considerações, análise de dados e discussão de resultados. A análise foi realizada com base neste estudo de campo e associada a pesquisa bibliográfica, que aborda conceitos da engenharia de software, tais como, ciclo de vida de um software, processo de software, modelos de processo de software, evolução de software, processo de manutenção e uma visão da tecnologia ITIL.

Palavras-Chave: processo de software; análise; pesquisa; qualidade de software; suporte; manutenção de software.

ABSTRACT

The work Process Analysis of Development and Support Services Software in businesses of development of Passo Fundo, presents itself initially a field study, in order to investigate how the process of software is managed and the process of maintaining in businesses of the city. Featuring, subsequently, a description of considerations, data analysis and discussion of results. The analysis was performed based on this field study and associated the research bibliography that approaches concepts of software engineering, such as, the life cycle of a software, process of software, process models of software, evolution of software, process of maintaining and a vision of technology ITIL.

Keywords: software process; analysis; research; software quality; support; maintenance software.

LISTA DE TABELAS

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O tema Análise do Processo de Desenvolvimento e Suporte a Serviços de Software nas empresas de desenvolvimento de Passo Fundo, busca possibilitar um estudo de como é gerido o processo de software e o processo de manutenção nas empresas entrevistadas.

O trabalho trata-se de um estudo bibliográfico abordando conceitos da engenharia de software, tais como, ciclo de vida de um software, processo de software, modelos de processo de software, evolução de software, processo de manutenção e uma visão da tecnologia ITIL.

Além disso, apresenta a avaliação de resultados de um estudo de campo, aplicado em forma de questionário, nas empresas de desenvolvimento de software de Passo Fundo.

Aliando estes dois fatores, referencial teórico e estudo de campo, foram realizadas considerações, discussões e estudos dos resultados, seguindo as metodologias de pesquisa determinadas e descritas no decorrer do projeto.

1.1 MOTIVAÇÃO

Chegou-se a este questionamento pelo fato de que até o momento não existiam informações e parâmetros documentados relacionados ao processo de desenvolvimento no mercado do município e qual o direcionamento deste mercado quanto a tecnologias e metodologias utilizadas.

A manutenção e evolução de software são parte do processo de desenvolvimento e a maneira como se desenvolve um software hoje, vai implicar diretamente em seus resultados e na sua posterior evolução, além disso, as mudanças são inevitáveis. Sendo assim, a maior parte dos custos de empresas de desenvolvimento é com a manutenção de seus softwares e não com o desenvolvimento de novos, sendo um motivo para a escolha deste tema.

Sendo assim, com esta análise poderão ser identificados problemas no processo de software e processo de manutenção, procurando encontrar os possíveis motivos e, com base em estudos de campo e do referencial teórico, tirar conclusões e considerações que possam aparecer como alternativas para resolução destes problemas encontrados.

Os principais questionamentos da pesquisa seriam: como se encontra, como é gerido e qual o direcionamento das empresas quanto ao processo de software em Passo Fundo? Como é realizado o suporte a serviços nestas empresas? Quais os principais problemas encontrados nas questões anteriores? De que maneira poderia ser resolvido?

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Analisar o Processo de Desenvolvimento e Suporte a Serviços de Software nas empresas de desenvolvimento de Passo Fundo.

1.2.2 Objetivos Específicos

- Realizar estudo bibliográfico abordando conteúdos relacionados ao tema;
- Realizar pesquisa de campo nas empresas de desenvolvimento de Passo Fundo;
- Obter conclusões e considerações sobre os resultados encontrados.

2 A ENGENHARIA DE SOFTWARE

O desenvolvimento de software é uma área que está em constante crescimento desde os primórdios da computação. Com o aumento do uso dos computadores na maioria das atividades cotidianas, aumentou também a demanda por soluções computadorizadas. Desta forma, o software elevou seu grau de importância nas diferentes áreas do conhecimento e a informação passa a ser a maior riqueza das empresas do mundo contemporâneo.

Com o software se tornando algo indispensável em nossas vidas afetando vários aspectos, difundindo-se em nosso comércio, em nossa cultura, em nossas atividades do dia-a-dia, surge a necessidade da criação de padrões e técnicas que agilizem, auxiliem e organizem este desenvolvimento, tudo objetivando suprir a necessidade de atingir bons níveis de qualidade de software, objetivo principal da engenharia de software.

Alguns fatores passam a ser levados em conta como o tempo gasto para a conclusão de um projeto, os custos, uma vez que, o processo de desenvolvimento precisa ser organizado, planejado e gerenciado, pois não é mais apropriada a idéia de construir software quase que de forma “artesanal” como construído na década de 1960. Esses fatores passaram a impulsionar a adoção de práticas de engenharia de software e, a partir disso, diversos autores passaram a tentar conceituá-la, iniciava então um novo contexto da criação de software.

A engenharia segundo definições do Software Engineering Institute (1997), é a aplicação sistemática do conhecimento cientifico na criação e construção de soluções para problemas práticos a serviço do gênero humano e a Engenharia de Software é uma forma de engenharia onde se aplicam conceitos da Ciência da Computação e da Matemática para alcançar soluções com melhor custo-benefício para o problema do software.

“Engenharia de software: (1) aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software, isto é, a aplicação da engenharia ao software. (2) O estudo de abordagens como as de (1)”. (IEEE *apud* PRESSMAN, 2010, p.17).

A engenharia de software está intimamente relacionada com a construção de software, posicionando-se como um processo que se preocupa com todas as fases do desenvolvimento de um software, desde a análise até sua afirmação, ou seja, por toda sua existência.

Segundo Pressman (2010), apesar de serem tantas as definições surgidas sobre engenharia de software, a definição mais coerente é uma primeira definição proposta por Fritz Bauer, apresentada na conferência pioneira sobre engenharia de software, que diz que: “A engenharia de software é a criação e a utilização de sólidos princípios de engenharia, a fim de

obter softwares econômicos que sejam confiáveis e que trabalhem eficientemente em máquinas reais". (BAUER *apud* PRESSMAN, 2010).

A engenharia de software surge como um meio organizador do desenvolvimento de software, sempre com o foco na qualidade do mesmo. Cabe a ela dizer qual a maneira mais eficaz de se desenvolver um software e a partir de seus conceitos, identificar o método mais apropriado para um determinado conjunto de circunstâncias que envolvem algum contexto.

Para que o gerente ou analista de software tenha maior facilidade e controle das atividades referentes ao desenvolvimento, Pressman (2010) organizou a engenharia de software em camadas, conforme a figura 1, sendo elas as seguintes:

- Métodos;
- Ferramentas;
- Procedimentos.

Figura 1 - Engenharia de Software em camadas.

Fonte: Pressman (2010, p.17).

Os procedimentos ou processos atuam como base da engenharia de software, pois são eles que definem a ordem em que cada atividade acontece, estabelecendo uma sequência de eventos, além de pensar qual será resultado ou retorno do sistema (documentos, modelos, dados, formulários, relatórios, entre outros), além das estratégias para se assegurar a qualidade e a escalabilidade do software em contexto, possibilitando aos gerentes de software maior precisão para acompanhar o desenvolvimento do seu projeto.

Os métodos fornecem embasamento técnico que define como fazer o software, envolvendo uma grande gama de tarefas como análise e recolhimento da informação, planejamento e estimativas, modelagem do projeto, execução de testes e manutenção.

Segundo Pressman (2010), os métodos de software repousam num conjunto de princípios básicos que regem cada área da tecnologia e incluem atividades de modelagem e outras técnicas descritivas.

Na ultima camada temos as ferramentas que servem como um apoio aos métodos e aos processos, este apoio pode ser automatizado ou semi-automatizado constituindo o que conhecesse por “engenharia de software apoiada por computador”, ou, ferramentas CASE (Computer-Aided Software Engineering).

Diante da realidade em que se encontra a área de desenvolvimento de software pode-se perceber que é quase que inviável trabalhar individualmente. As pessoas necessitam trabalhar em equipes e seus esforços têm que ser concentrados, planejados e coordenados para que se venha a ter qualidade na produção de um determinado produto e para que se possa inibir o desperdício de tempo e dinheiro.

A engenharia de software com todos os seus processos, métodos e fundamentos motiva os engenheiros e desenvolvedores a utilizarem-se dela para construção de seus projetos. É possível notar claramente, conforme já foi citado, que todas as maneiras propostas para se criar software e todas as técnicas, mesmo cada uma tendo suas características e particularidades, objetivam encontrar a qualidade de software.

Mas apesar de tudo, em qualquer ramo do conhecimento, depende-se da disciplina e comprometimento de quem aplica a metodologia.

Atualmente, a engenharia de software dispõe-se de uma grande quantidade de fundamentos científicos, tanto teóricos, quanto práticos que possibilitam de maneira eficaz planejar, modelar, implantar e manter sistemas mantendo sempre o foco na qualidade deste produto. Desta forma, segundo Falbo (2005), existem algumas tarefas básicas que compõe a engenharia de software:

- Planejamento e Estimativa de Projeto;
- Análise e Requisitos de Software;
- Projeto da Estrutura de Dados;
- Arquitetura de Programa e Algoritmo de Processamento;
- Codificação;
- Teste e Manutenção.

2.1 TAREFAS BÁSICAS QUE COMPÕEM A ENGENHARIA DE SOFTWARE

O ciclo de vida de um software descreve as fases pelas quais o software passa desde a sua concepção até ele ficar inutilizável, sendo um importante e decisivo componente no processo de desenvolvimento.

Segundo Cordeiro (2006), ciclos de vida do software descrevem como um software deve ser desenvolvido. Basicamente definem a ordem global das atividades envolvidas em um contexto de projeto de software e propõe uma estratégia de desenvolvimento que pode ser aplicada a um determinado contexto de projeto de software.

Ele é uma visão em longo prazo do software que envolve um conjunto de atividades desde a concepção, desenvolvimento e o processo de manutenção após a sua entrada em operação até sua total obsolescência.

Existem três objetivos que estimulam a existência dos ciclos de vida:

- Definir as atividades a serem executadas em um projeto;
- Introduzir a coerência entre muitos projetos na mesma organização;
- Fornece pontos de checagem para controle de gerência e pontos de checagem para a decisão, ajudam o gerente a saber aonde ir e aonde não ir.

Sobre modelos de ciclo de vida Falbo (2005, p. 6), diz que:

> Um modelo de ciclo de vida pode ser entendido como passos ou atividades que devem ser executados durante um projeto. Para a definição completa do processo, a cada atividade, devem ser associados técnicas, ferramentas e critérios de qualidade, entre outros, formando uma base sólida para o desenvolvimento. Adicionalmente, outras atividades tipicamente de cunho gerencial, devem ser definidas, entre elas atividade de gerência e de controle e garantia da qualidade.

Segundo Falbo (2005), de maneira geral, conforme foi listado anteriormente, o ciclo de vida de um software envolve as seguintes fases (Figura 2):

- **Planejamento:** O objetivo do planejamento de projeto é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Uma vez estabelecido o escopo de software, uma proposta de desenvolvimento deve ser elaborada, isto é, um plano de projeto deve ser elaborado configurando o processo a ser utilizado no desenvolvimento de

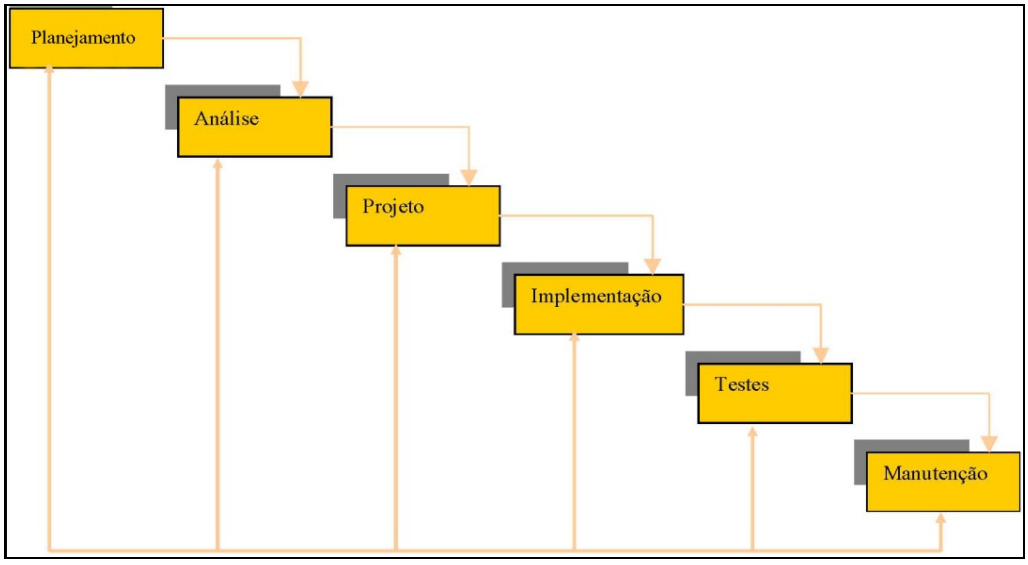
software. À medida que o projeto progride, o planejamento deve ser detalhado e atualizado regularmente. Pelo menos ao final de cada uma das fases do desenvolvimento (análise e especificação de requisitos, projeto, implementação e testes), o planejamento como um todo deve ser revisto e o planejamento da etapa seguinte deve ser detalhado. O planejamento e o acompanhamento do progresso fazem parte do processo de gerência de projeto.

- **Análise e Especificação de Requisitos:** Nesta fase, o processo de levantamento de requisitos é intensificado. O escopo deve ser refinado e os requisitos identificados. Para entender a natureza do software a ser construído, o engenheiro de software tem de compreender o domínio do problema, bem como a funcionalidade e o comportamento esperados. Uma vez identificados os requisitos do sistema a serem desenvolvidos, estes devem ser modelados, avaliados e documentados. Uma parte vital desta fase é a construção de um modelo descrevendo o que o software tem de fazer.
- **Projeto:** Esta fase é responsável por incorporar requisitos tecnológicos aos requisitos essenciais do sistema, modelados na fase anterior e, portanto, requer que a plataforma de implementação já esteja definida e seja bem conhecida entre os membros da equipe. Basicamente, envolve duas grandes etapas: projeto da arquitetura do sistema e projeto detalhado. A primeira etapa define a arquitetura geral do software, tendo por base o modelo construído na fase de análise de requisitos. Esta arquitetura deve descrever a estrutura de nível mais alto da aplicação e identificar seus principais componentes. O propósito do projeto é detalhar o processo de software, a partir dos componentes identificados na etapa anterior. Os componentes de software devem ser sucessivamente refinados em níveis de maior detalhamento, até que possam ser codificados e testados.
- **Implementação:** O projeto deve ser traduzido para uma forma passível de execução pela máquina. A fase de implementação realiza esta tarefa, isto é, cada unidade de software do projeto detalhado é implementada.
- **Testes:** inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os

diversos componentes devem ser integrados sucessivamente até se obter o sistema. Finalmente, o sistema como um todo deve ser testado.

- **Entrega e Implantação:** uma vez testado, o software deve ser colocado em produção. Para tal, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta fase é estabelecer que o software satisfaça os requisitos dos usuários. Isto é feito instalando o software e conduzindo testes de aceitação (validação). Quando o software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.
- **Operação:** nesta fase, o software é utilizado pelos usuários no ambiente de produção.
- **Manutenção:** Indiscutivelmente, o software sofrerá mudanças após ter sido entregue para o usuário. Alterações ocorrerão se erros forem encontrados, porque o software precisa ser adaptado para acomodar mudanças em seu ambiente externo, ou porque o cliente necessita de funcionalidade adicional ou aumento de desempenho. Muitas vezes, dependendo do tipo e porte da manutenção necessária, essa fase pode requerer a definição de um novo processo, onde cada uma das fases precedentes é reaplicada no contexto de um software existente ao invés de um novo.

Figura 2 - Definição do ciclo de vida de software.



Fonte: Repositório digital de imagens da Google (2012).

2.2 PROCESSOS DE SOFTWARE

Para melhorar a organização das tarefas de desenvolvimento a engenharia de software se baseia em processos. A qualidade do software esta intimamente ligada a qualidade dos mesmos, considerando que estes estejam claramente definidos.

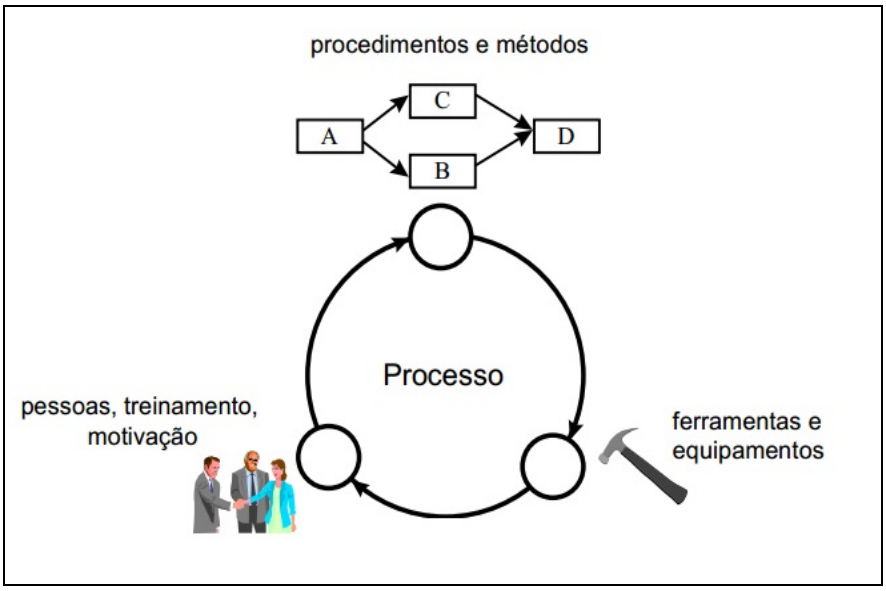
“Um processo de software é um conjunto de atividades relacionadas que levam a produção de um produto de software.” (SOMMERVILLE, 2011, p.18).

O processo de software funciona como um roteiro formado por uma série de passos previstos, que de forma organizada através de métodos, práticas e outras atividades visam criar a custo de tempo um resultado de alta qualidade.

Para se definir um processo de software é necessário que se tenha pronta toda documentação e especificação detalhada do que se quer desenvolver. A partir disso, podem-se definir quais serão os passos do desenvolvimento, qual o tempo estimado para cada processo, quem realizará cada processo e quais os custos de cada processo para o projeto total, assim pode-se vislumbrar de forma clara e organizada quais realmente são os resultados esperados.

Segundo o Software Engineering Institute (1997), um processo de software pode ser melhorado potencializando as pessoas que fazem parte dele, de forma organizada e produtiva. Além de que um processo de software efetivo significa que pessoas, métodos, e tecnologia formam um todo integrado (Figura 3).

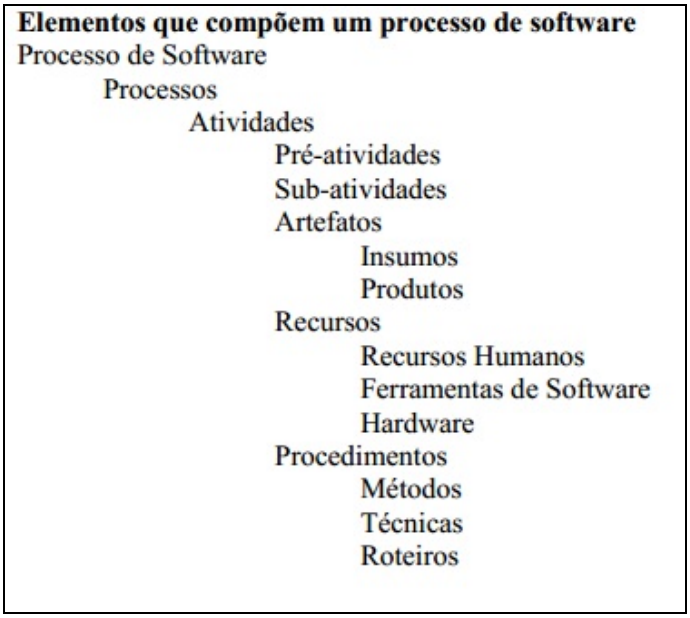
Figura 3 - Formação de um processo de software.



Fonte: Repositório digital de imagens da Google (2012).

"Um processo eficaz deve, claramente, considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários e a habilidade, o treinamento e a motivação do pessoal envolvido." (FALBO, 2005, p.5).

Figura 4 - Elementos que compõe um processo de software.



Fonte: Falbo (2005, p.5).

Os passos para criação de um processo de software dependem totalmente do tipo de software que esta sendo construindo, sendo que um processo que agora é adequado para um determinado caso, pode não ser adequado para outro e futuramente pode não ser o melhor para um software da mesma natureza.

Segundo Falbo (2005), os processos podem ser classificados segundo seu propósito em:

- Atividades de Desenvolvimento (ou Técnicas de Construção);
- Atividades de Gerência;
- Atividades de Garantia da Qualidade.

As Atividades de Desenvolvimento são aquelas que tocam diretamente no desenvolvimento do produto de software a ser entregue a um cliente. Exemplos: Especificação e análise de requisitos, projeto e implementação.

Na sequência temos as Atividades de Gerência, que como o próprio nome diz referem-se às atividades gerenciais do projeto relacionadas ao planejamento e acompanhamento, tais

como realização de estimativas, construção de cronogramas, análise dos riscos do projeto entre outros.

Já as Atividades de Garantia e Qualidade atendem as tarefas que dizem respeito à garantia da qualidade do produto em desenvolvimento e do processo de software utilizado nas etapas de desenvolvimento como, por exemplo, revisões e inspeções de produtos (intermediários ou finais) do desenvolvimento.

Como se pode perceber, a engenharia de software gira em torno de processos e produtos. Os processos visam ordenar e possibilitar controle sobre as atividades do projeto e os produtos aparecem como o resultado obtido a partir da aplicação de uma série de processos em um modelo especifico.

Conforme Sommerville (2011), existem diversos processos de software diferentes, mas todos devem conter quatro atividades fundamentais comuns:

- *Especificação de software:* são especificadas as funcionalidades e restrições do software;
- *Projeto e Implementação de Software:* o software deve ser produzido seguindo a especificação;
- *Validação de Software:* o software deve ser validado antes de ser entregue ao cliente, sempre procurando garantir que atenda as suas especificações;
- *Evolução de software:* para atender às necessidades mutáveis do cliente, o software deve evoluir.

As atividades dos processos podem variar de um projeto para o outro em seu nível de detalhamento, nos prazos de realização e também nos resultados. Como já foi dito, pode-se ser usado diversos tipos de processos para construção de um tipo de produto, sendo que, se for utilizado um processo inadequado para determinada situação com certeza teremos perca na qualidade do produto e problemas no desenvolvimento do projeto.

São alguns fatores que influenciam a definição de um processo, segundo Falbo (2005):

- Tipo de Software (sistema de informação, sistema de tempo real, etc.);
- Paradigma (estruturado, orientado a objetos);
- Domínio da Aplicação;
- Tamanho e Complexidade;

- Características da Equipe;
- Entre outros.

"Não existe um processo ideal, a maioria das organizações cria seus próprios processos de desenvolvimento de software." (SOMMERVILLE, 2011, p.19).

A definição do processo de software é uma tarefa de grande complexidade, pois envolve recursos, ferramentas, tempo, custo, risco, e, além disso, a tomada de decisões sobre seu direcionamento durante a execução das atividades depende das pessoas envolvidas no projeto, tanto clientes, quanto desenvolvedores e gerentes. Por este motivo, o processo de software está em constante evolução e deve estar aberto a transformações, para adequar-se de forma que venha sempre tirar o melhor rendimento das pessoas que compõe a equipe e das características do software a ser desenvolvido.

Conforme Sommerville (2011), as organizações que apresentam menor diversidade de processos, podem melhorá-los utilizando a padronização. Ela vai possibilitar melhor comunicação, reduzindo o tempo destinado a treinamento, além de tornar mais econômico o apoio ao processo automatizado.

Como tudo o que acontece no processo de desenvolvimento influi diretamente no produto final, quando temos processos escaláveis e modulares teremos um software com as mesmas características; possibilitando sempre a extensão e o crescimento do mesmo.

Como sempre temos os processos envolvidos a elaboração de um produto, muitos autores referenciam o conjunto de processos como sendo um "ciclo de vida" que vai desde a concepção do projeto, passa pela sua implementação, entrega, utilização e manutenção, enfim por toda a vida do software.

2.3 MODELOS DE PROCESSO DE SOFTWARE

Os modelos de processo de software são esquemas criados para gerar uma estrutura útil às atividades relacionadas a desenvolvimento de software, além de servir como se fosse um roteiro para as equipes de desenvolvimento.

Modelo de processo é uma representação simplificada e abstrata de um determinado processo de software que esta sendo descrito, são estes modelos que ajudam a explicar as diferentes abordagens do desenvolvimento de software.

Conforme Sommerville (2011), cada tipo de modelo diferente representa uma visão particular de um processo e fornece informações parciais sobre ele. Enfatizando que um

modelo pode mostrar as atividades e sua sequência, porém, não necessita obrigatoriamente informar os papéis das pessoas envolvidas.

Sommerville (2011), ainda coloca que podemos ver a estrutura de um determinado modelo de processo como um *framework* do processo, onde os detalhes das atividades específicas ficam abstratos.

Um modelo de ciclo de vida de software é simplesmente um método para organizar os diversos componentes e atividades durante o desenvolvimento de um sistema.

Segundo Pressman (2010), ele surge como uma estratégia de desenvolvimento que envolve todas as camadas de processo, métodos, ferramentas e fases genéricas.

O objetivo principal dos modelos de processos de software, conforme Sommerville (2011), é descrever as transações entre as etapas, decidir e prover critérios para selecionar atividades que devem ser desenvolvidas em cada momento.

Existem diversos modelos de processos cada um com suas particularidades e que podem ser aplicados em diferentes contextos, tudo depende da especificação de cada projeto, a partir de onde um gerente de projetos vai julgar qual modelo utilizar. Desta forma, não existe um modelo melhor que outro ou mais eficiente, cada empresa deve saber qual modelo se adéqua com maior consistência em sua realidade, sua forma de trabalho e aos projetos com que trabalha.

McDonald e Welland (2001), dizem que a uma grande quantidade de desenvolvedores não utiliza um processo de desenvolvimento bem definido e documentado. Os autores salientam também que alguns utilizam modelos padronizados pela comunidade de software e outros adaptam os modelos criados dentro da organização à sua maneira.

Segundo Pressman (2010), os modelos de processos de software se dividem em: Modelos tradicionais e Modelos Ágeis.

2.3.1 Modelos Tradicionais

Os modelos de processo de desenvolvimento considerados tradicionais baseiam-se todos no modelo Cascata, o qual sugere que a construção do sistema seja feita linearmente seguindo uma sequência de fases.

Segundo Rabello e Bortoli (2006), por apresentarem-se como modelos de processos pioneiros, os modelos tradicionais, ou ainda chamados de modelos convencionais, trazem uma estrutura útil para as atividades de engenharia de software e para os modelos que foram

surgindo depois deles, funcionando como um roteiro efetivo para os profissionais ligados a desenvolvimento de software.

Nesta sessão, serão apresentadas as características principais, os objetivos e a forma de trabalho dos modelos de processo tradicionais.

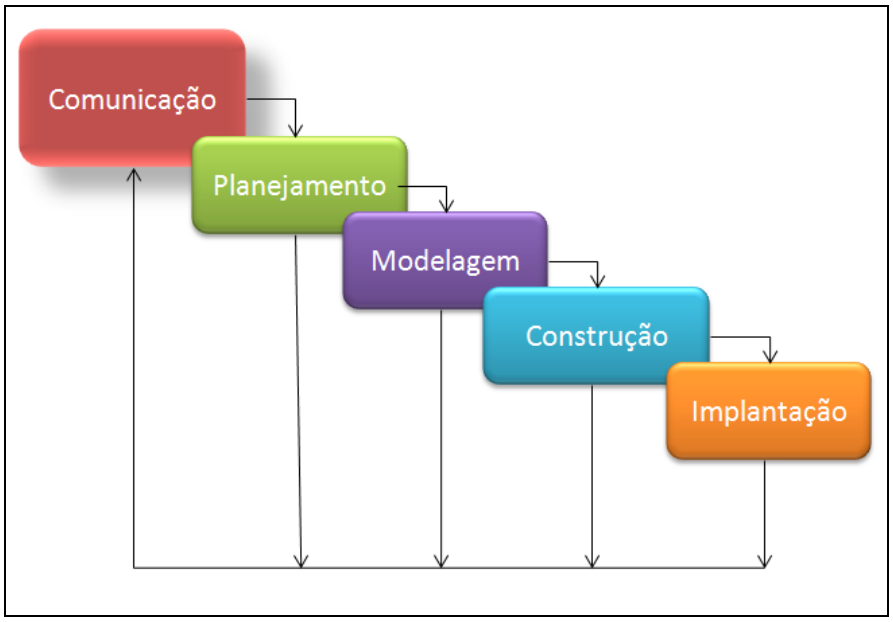
Entre modelos de processo considerados tradicionais, que serão abordados, tem-se:

- Modelo Cascata;
- Modelo Incremental;
- Modelos Evolucionários: Prototipagem e Espiral;
- Modelo RUP.

2.3.1.1 Modelo Cascata

O modelo cascata é considerado o modelo base para todos os outros modelos de processos existentes. A estrutura de organização deste modelo se assemelha com o conceito de ciclo de vida do software, e é extremamente linear (Figura 5).

Figura 5 - Modelo Cascata.



Fonte: Repositório digital de imagens da Google (2013).

Segundo Rabello e Bortoli (2006), o modelo cascata ou clássico, como também é conhecido, foi proposto por Royce em 1970, e foi o único modelo com aceitação geral até meados da década de 1980, caracterizando-se como um método linear e sequencial.

“O primeiro modelo do processo de desenvolvimento de software a ser publicado foi derivado de processos mais gerais da engenharia de sistemas.” (ROYCE *apud* SOMMERVILLE, 2011, p.20).

É um modelo de processos indicado quando os requisitos do problema foram bem especificados, são bem definidos e são estáveis, onde as possíveis futuras adaptações sejam bem definidas e os aperfeiçoamentos calculados.

“A proposta do modelo Cascata Puro consiste na execução das atividades de desenvolvimento de software em uma sequência ordenada.” (CORDEIRO, 2006, p.1).

Desta forma, existe certo encadeamento entre as etapas, assim uma etapa só vai se iniciar quando a etapa anterior for concluída e o resultado de cada fase será a aprovação de um ou mais documentos.

A documentação pode ser composta por manuais de software, que se bem elaborados, podem servir como referência no desenvolvimento e andamento dos projetos.

Rabello e Bortoli (2006) colocam que estes manuais de software podem ser divididos em operação (funcionamento do software), manutenção (limitações de rede, hardware e as condições ideais para o bom funcionamento) e sistema (orientar todos os envolvidos no sistema).

Segundo Sommerville (2011), durante a execução do projeto serão encontrados problemas em requisitos, na codificação, no projeto e assim por diante, o que comprova que o processo de software não é algo totalmente linear, mas exige de qualquer maneira o *feedback* de uma fase para a outra. Assim as etapas podem se repetir, os documentos podem ser modificados e as alterações realizadas em cada etapa.

Desta forma, teremos o modelo Cascata com possibilidade à realimentação, onde é possível permitir que em fases posteriores, seja realizada a revisão e alteração de resultados obtidos em fases anteriores.

“A realimentação entre as fases possibilita a correção de erros à medida que vão sendo descobertos.” (RABELLO E BORTOLI, 2006, p.21).

Entre as vantagens do modelo Cascata podemos destacar:

- Facilidade no gerenciamento;

- Desenvolvimento estruturado com sequência e ordem nas etapas, onde todas as atividades são relevantes;
- Identifica um conjunto fixo de documentação, produzida como resultado de cada etapa.

Porém, o modelo também apresenta algumas limitações que devem ser ressaltadas:

- Modelo um tanto engessado quanto a divisão dos estágios, o que dificulta alterações no processo no decorrer do desenvolvimento;
- Apenas o gerente do projeto tem uma visão global do sistema;
- Nem sempre será possível seguir o fluxo sequencial proposto pelo modelo;
- A dificuldade do cliente na especificação dos requisitos pode ser um problema no inicio de qualquer projeto;
- O cliente somente terá contato com o software quando ele estiver concluído;
- A realimentação entre as etapas dificulta a gerência do projeto.

"Este modelo tem um papel importante com os trabalhos da engenharia de software porque produz um padrão no qual se encaixam os métodos para análise, projeto, codificação e manutenção." (RABELLO E BORTOLI, 2006, p.24).

Apesar de suas limitações e de seu caráter burocrático, o modelo cascata é um modelo muito importante, uma vez que é consistente com outros modelos de processos da engenharia e a documentação é gerada em cada fase do desenvolvimento.

2.3.1.2 Modelo Incremental

Segundo Sommerville (2011), o Modelo Incremental foi proposto por Mills na década de 1980 e tem a finalidade de proporcionar um *feedback* mais rápido ao cliente.

Conforme Rabello e Bortoli (2006), esta abordagem proporciona aos clientes a possibilidade de experimentar versões operacionais do sistema, as vezes mesmo que este não esteja totalmente terminado, facilitando o esclarecimento dos requisitos para incrementos seguintes e para versões posteriores à atual, reduzindo, assim o retrabalho no processo de desenvolvimento.

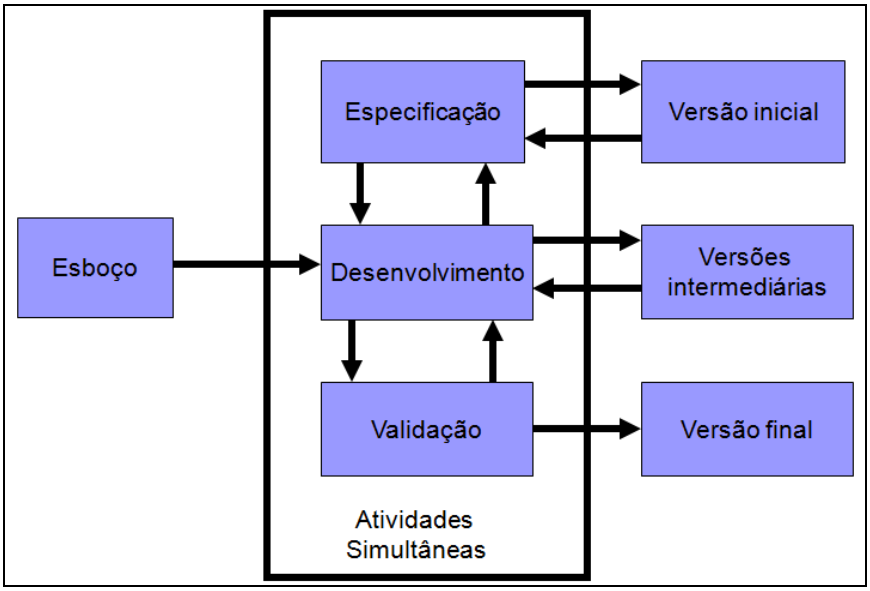
“O modelo incremental como um processo de ciclo de vida iterativo e incremental com características do ciclo de vida sequencial.” (TRAMMELL *apud* BERTHOLDO E BARBAN, 2010, p.3).

“O modelo incremental combina elementos do modelo em cascata aplicado de maneira iterativa.” (PRESSMAN, 2010, p.40).

Desta forma, a cada incremento finalizado, temos uma nova versão ou uma nova funcionalidade do sistema que pode ser repassada ao cliente.

Apesar de combinar elementos do modelo cascata dentro de cada um dos seus incrementos, o modelo incremental não é totalmente linear e sequencial, pois os processos de especificação, desenvolvimento e validação, realizados em cada incremento, podem ser intercalados, sem haver separação alguma, podendo acontecer até mesmo simultaneamente (Figura 6).

Figura 6 - Modelo Incremental.



Fonte: Sommerville (2011, p.31).

Segundo Sommerville (2011), neste tipo de processo de desenvolvimento, primeiramente é criado um esboço, juntamente com o cliente, com as principais funcionalidades do sistema, julgando, conforme sua necessidade, quais são mais e menos importantes, a partir daí inicia-se o desenvolvimento seguindo uma série de estágios (ordem dos incrementos), conforme as prioridades estabelecidas pelo cliente. Cada incremento irá corresponder uma funcionalidade.

É sempre necessário que se tenha um bom entendimento e definição do escopo do sistema antes de iniciar o desenvolvimento para evitar inconformidades futuras.

“Quando um modelo incremental é usado, o primeiro incremento é frequentemente chamado de núcleo do produto” (PRESSMAN, 2010, p.40). O núcleo do produto serve de base para os outros incrementos.

Conforme Rabello e Bortoli (2006), assim que forem definidos os incrementos, eles serão definidos e detalhados conforme as funcionalidades do sistema, então este incremento é iniciado utilizando o processo de desenvolvimento mais adequado.

Depois de definido e iniciado o desenvolvimento de um determinado incremento, não podem ser realizadas mudanças durante as atividades deste incremento.

“Os requisitos são congelados, ou seja, não são aceitas mudanças de requisitos no estágio atual, embora possam ser previstas evoluções nos requisitos para os próximos incrementos.” (RABELLO E BORTOLI, 2006, p.32).

O modelo de desenvolvimento incremental apresenta alguns pontos fortes relevantes, nos quais podemos destacar:

- É um modelo de desenvolvimento interativo, portanto, o cliente não precisa esperar que o sistema todo fique pronto, para poder ter contato com o produto;
- Os incrementos iniciais são como protótipos que auxiliam na especificação de requisitos para os incrementos posteriores;
- Embora apareçam problemas em alguns incrementos, temos a possibilidade de que sejam corrigidos em incrementos posteriores, o que reduz o risco de o projeto como um todo fracassar;
- Todos os incrementos passam por testes e validação, o que reduz a possibilidade de falhas;
- Reduz o tempo de desenvolvimento.

Porém, Sommerville (2011), destaca alguns fatores que aparecem como problemas deste tipo de abordagem:

- Os incrementos não podem ter mais do que vinte mil linhas de código cada um;
- Todo incremento deve produzir uma funcionalidade para o sistema;
- Dificuldade para identificar facilidades comuns para todos os incrementos.

“O desenvolvimento incremental é particularmente útil quando não há mão de obra disponível para uma implementação completa, dentro do prazo comercial de entrega estabelecido para o projeto.” (PRESSMAN, 2010, p.40).

Este modelo aparece como base para as metodologias ágeis de desenvolvimento. Conforme Rabello e Bortoli (2006), o modelo ágil *Extreme Programming* (XP), é considerado a evolução mais recente do modelo incremental.

2.3.1.3 Prototipagem

O modelo de prototipagem se mostra com uma visão evolutiva do desenvolvimento do software, que a partir de versões iniciais do software (protótipos) desenvolve o sistema como um todo.

“Um protótipo é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções.” (SOMMERVILLE, 2011, p.30).

Como muitos fatores podem ser testados nas versões iniciais e já teremos uma idéia de como o software ira se comportar de forma global, o modelo de prototipagem é muito útil para prever mudanças, prover soluções a problemas e possibilitar que os custos de desenvolvimento sejam calculados antecipadamente.

Segundo Pressman (2010), o protótipo serve como um mecanismo para identificar requisitos do software e que apesar de a prototipagem ser usada como um modelo de processos independente, ela é normalmente utilizada como um recurso ou técnica que pode ser implementada no contexto de outros modelos.

Por ser uma abordagem interativa, a prototipagem permite ao usuário final adquirir experiência práticas com o sistema antes de ele ser desenvolvido. Muitas vezes isso possibilita que possa ser definida a aparência externa do sistema, por exemplo, telas, relatórios e diálogos.

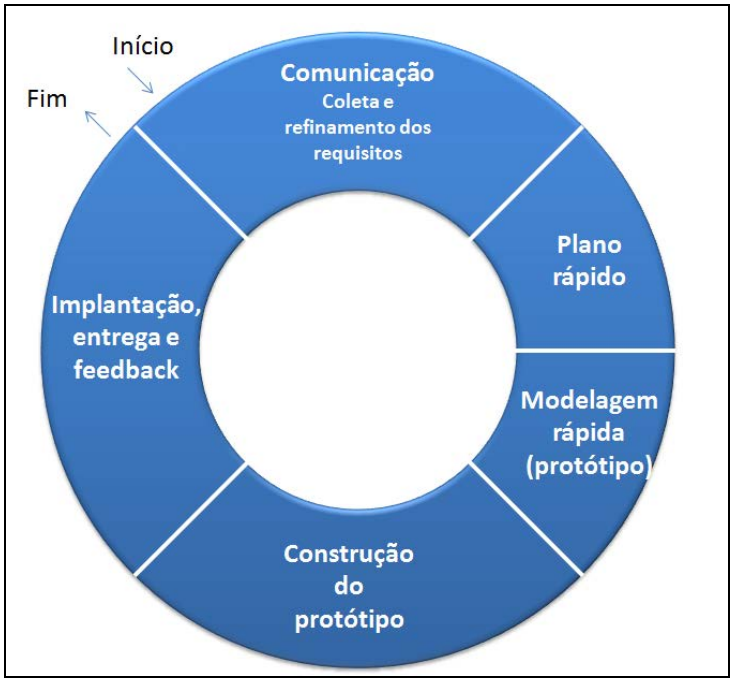
Segundo Pfleeger (2004), depois de o cliente e usuários definirem um conjunto simples de requisitos já pode ser iniciado o desenvolvimento de um “projeto rápido” ou “protótipo”, a partir daí, as alternativas para o produto final começam a surgir.

Examinam-se as telas, tabelas, relatórios e outras saídas do sistema, diretamente utilizadas pelos clientes e usuários. Assim que os usuários e clientes decidem o que realmente

querem, os requisitos são revisados. Uma vez que haja consenso de como deveriam ser os requisitos, o desenvolvedor se volta para as atividades de requisitos, a fim de reconsiderar e alterar a especificação, enfim, o projeto pode ser codificado em sua versão global.

Conforme Rabello e Bortoli (2006), as etapas do processo da prototipagem interagem entre si e a cada refinamento do protótipo é realizada uma revisão dos requisitos e em cada revisão são percorridas todas as etapas (Figura 7).

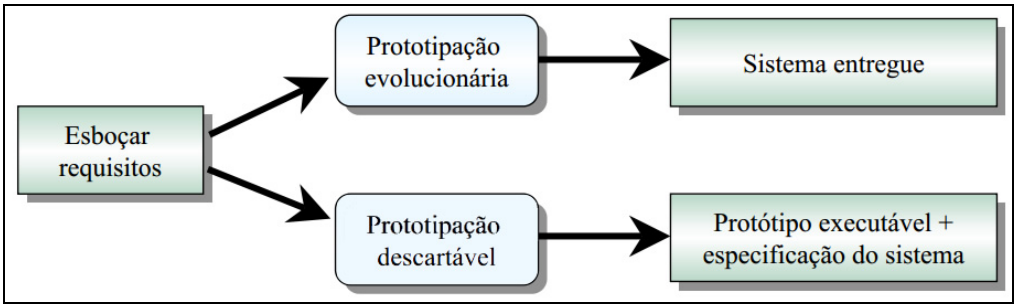
Figura 7 - Prototipação.



Fonte: Repositório digital de imagens da Google (2013).

A prototipação pode ser dividida em dois tipos: evolucionária e descartável (Figura 8).

Figura 8 - Prototipação evolucionária e descartável.



Fonte: Rabello e Bortoli (2006, p.27).

A prototipação evolucionária tem a finalidade de gerar um sistema funcional para o usuário final, ou seja, um protótipo inicial é produzido e refinado em várias etapas de avaliação e alterações até torna-se um produto final.

Segundo Sommerville (2011), neste tipo de prototipação o desenvolvimento começa a partir dos requisitos melhor compreendidos e de maior prioridade, sendo que, os requisitos de menor prioridade só serão incorporados caso o cliente solicite.

“A prototipagem evolucionária pode ser utilizada para o desenvolvimento de *sites*, e aplicações de comércio eletrônico.” (RABELLO E BORTOLI, 2006, p.27).

É um tipo de prototipagem indicado para sistemas de pequeno e médio porte, onde seja necessário o rápido fornecimento de resultados, o que por consequência requer grande envolvimento e compromisso do usuário com o sistema.

Temos ainda a prototipação descartável que tem como resultados a especificação dos requisitos. Consiste na criação de um protótipo descartável de software, que é desenvolvido rapidamente para que usuários e clientes possam fornecer *feedback* da especificação. Uma vez realizada a especificação, o protótipo não é mais útil e é descartado.

“A prototipagem descartável auxilia no aprimoramento e na classificação da especificação do sistema.” (RABELLO E BORTOLLI, 2006, p.28).

2.3.1.4 Modelo Espiral

Conforme Pressman (2010), o modelo de processos em espiral foi criado por Barry Boehm em seu artigo de 1988 *A Spiral Model of Software Development and Enhancement*, que agrega a natureza interativa da prototipagem com os aspectos burocráticos do modelo cascata.

Segundo Sommerville (2011), é um modelo evolucionário dirigido aos riscos, onde o processo de software é representado como uma espiral e não como uma sequência de atividades lineares com alguns retornos de uma etapa para a outra, sendo que, cada volta na espiral corresponde a uma fase do processo de software.

Nesta metodologia, cada fase ou etapa do processo é precedida por uma análise dos riscos e sua execução é feita de forma evolutiva, portanto, alterações que forem necessárias durante o desenvolvimento, só poderão ser solucionadas na próxima etapa.

“O objetivo do modelo evolucionário é manipular da melhor forma um conjunto de requisitos incertos ou sujeitos a alterações.” (RABELLO E BORTOLI, 2006, p.33).

A organização e a forma como as coisas acontecem neste modelo de processos permite que ele seja adaptado para uma aplicação ao longo da vida do software, uma vez que ele é diferente de outros modelos de processo que terminam quando o software é entregue.

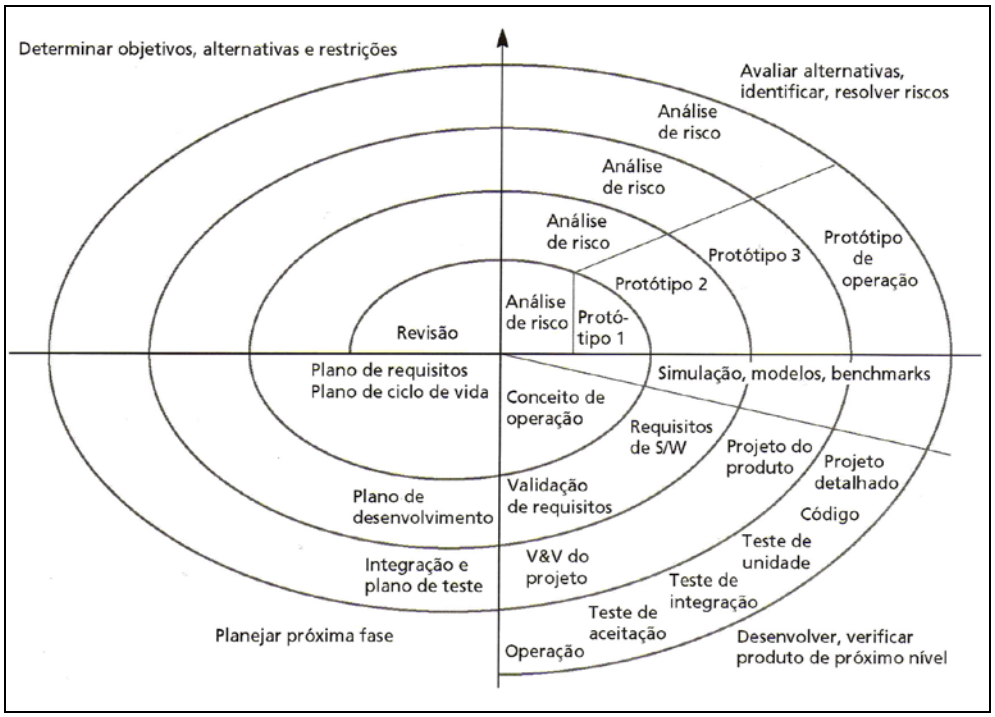
Conforme Rabello e Bortoli (2006), o modelo espiral oferece um meta-modelo que pode acomodar diversos modelos específicos, conforme a necessidade de cada projeto.

Desta forma, é possível somar a sua estrutura características e particularidades de outras metodologias.

Segundo Pressman (2010), o desenvolvimento em espiral, em sua essência, busca unir as melhores características do modelo cascata e prototipação, agregando um novo elemento: a análise dos riscos.

Conforme podemos ver na Figura 9, o modelo se divide em quatro atividades principais que são representadas na forma de quadrantes.

Figura 9 - Modelo Espiral de Boehm.



Fonte: Sommerville (2011, p.33).

Segundo Sommerville (2011), cada um dos quadrantes ou regiões dentro da espiral é composto por uma série de atividades, sendo que estes quadrantes remetem ao seguinte:

- 1º Região (Definição de objetivos): são definidos os objetivos específicos do projeto, identificadas as restrições ao processo e ao produto, e é gerado um

plano de gerenciamento; além de serem identificados os riscos do projeto e elaboradas estratégias e soluções para passar por estes riscos sem problemas.

- 2º Região (Avaliação e Redução dos Riscos): para cada risco identificado anteriormente é realizada uma análise e proposta uma solução. Neste quadrante são tomadas as medidas para redução dos riscos.
- 3º Região (Desenvolvimento e Validação): terminada a análise dos riscos, já se pode definir um modelo de desenvolvimento mais adequado para construção do sistema.
- 4º Região (Planejamento): o projeto e todas as atividades realizadas na etapa atual são revisados, desta forma, são tomadas decisões que se referem a continuidade do modelo com mais uma volta na espiral, então são elaborados os planos para próxima fase do projeto.

É possível notar que o principal diferencial do modelo espiral em relação a outras metodologias é a alta preocupação com os riscos, evidenciando o valor que este modelo tem dentro da engenharia de software, pois, a redução de riscos é um fator muito importante quando pensamos em gerenciamento de processos.

Sommerville (2011), define três tipos de riscos: riscos relacionados ao projeto, riscos relacionados ao produto e riscos relacionados ao negócio, podemos ver alguns exemplos na Tabela 1.

Tabela 1 – Riscos de Software

Risco	Tipo de Risco	Descrição
Rotatividade de pessoal	Projeto	O pessoal experiente abandona o projeto antes do término.
Mudança de Gerenciamento	Projeto	Mudança no gerenciamento com a definição de diferentes prioridades.
Indisponibilidade de hardware	Projeto	Hardware essencial ao projeto não será entregue dentro do prazo.
Alteração nos requisitos	Projeto e Produto	Maior número de mudanças no projeto do que o previsto.
Atrasos na especificação	Projeto e Produto	As especificações de interfaces essenciais não disponíveis dentro dos prazos.
Tamanho subestimado	Projeto e Produto	O tamanho de sistema foi subestimado.
Baixo desempenho de ferramentas Case	Produto	As ferramentas Case que apóiam o desenvolvimento do projeto não apresentam desempenho conforme o previsto.
Mudanças na tecnologia	Negócio	A tecnologia em que o projeto está sendo

		desenvolvido é superada por nova tecnologia.
Concorrência com o produto	Negócio	Um produto concorrente é lançado no mercado antes da conclusão do sistema.

Fonte: Rabello e Bortoli (2006, p.36).

"Identificar os riscos é uma maneira de detectar as ameaças ao projeto, tais como estimativas, cronogramas, recursos, entre outros." (RABELLO E BORTOLI, 2006, p. 35).

2.3.1.5 Modelo RUP

O *Rational Unified Process* (RUP) é considerado um modelo de processos moderno e híbrido, pois procura agrupar e apoiar-se nos melhores recursos e características dos modelos convencionais. Acrescentando ainda a notação da UML (*Unified Modeling Language*) é um processo de engenharia de software criado para apoiar o desenvolvimento orientado a objetos.

"Ele reúne elementos de todos os modelos de processos genéricos, ilustra boas práticas, e no projeto e apóia a prototipação e a entrega incremental." (SOMMERVILLE, 2011, p.34).

Segundo Rabello e Bortoli (2006), o RUP é orientado a casos de uso, centrado na arquitetura e interativo. Orientado a casos de uso no sentido de que esses casos são utilizados para estabelecer como será o comportamento do sistema, estabelecendo as relações e comunicações entre os participantes do projeto. É centrado na arquitetura, pois a arquitetura do sistema é o principal fator para entendimento do contexto, construção, gerenciamento, evolução do sistema a ser desenvolvido. E incremental pois envolve a integração continua em versões, de forma que cada versão incorpore mudanças incrementais conforme forem necessárias.

O RUP define perfeitamente quem é responsável pelo que, como as coisas deverão ser feitas e quando devem ser realizadas, descrevendo todas as metas de desenvolvimento especificamente para que sejam alcançadas, por este motivo, é considerado um modelo bastante burocrático, que apesar de poder ser adaptado a projetos de qualquer proporção, é mais utilizado na construção de sistemas maiores e mais complexos.

Conforme Pressman (2010), esta metodologia gera uma série de modelos e documentos, como consequência da relação com a UML, porém, por muitas vezes, a criação desta documentação não é seguida a risca por engenheiros de software com a finalidade de tornar o desenvolvimento mais ágil e produtivo.

Sommerville (2011), descreve o RUP seguindo três perspectivas, que geralmente vão aparecer combinadas em um único diagrama:

- Uma perspectiva dinâmica, que mostra as fases do modelo ao longo do tempo;
- Uma perspectiva estática, que mostra as atividades realizadas no processo;
- Uma perspectiva prática, que sugere boas práticas a serem usadas durante o processo.

Dentro da sua capacidade de ser um modelo híbrido, que agrupa várias visões e técnicas diferentes, o RUP ressalta duas características: análise dos riscos e modelagem baseada na arquitetura do sistema.

Conforme Pressman (2010), como acontece em alguns outros modelos convencionais, o RUP é composto por fases, sendo quatro fases distintas, são elas:

- **Concepção:** fase de entendimento das necessidades e visão do projeto. São definidos todos os casos de uso e a descrição dos mesmos, definidos as entidades externas (atores), abrangendo as atividades de comunicação com o cliente e de planejamento do negócio que contém a análise de riscos, estimativas de recursos e prazos.
- **Elaboração:** atividades de modelagem do modelo genérico de processo. Nesta fase acontece a especificação dos fatores principais do sistema e refina os casos de uso surgidos na concepção. Como resultado desta etapa teremos o plano do projeto, estimativa de custos, o cronograma definido, os objetivos estarão visíveis, serão estabelecidos soluções para os riscos e será possível escolher a arquitetura mais apropriada para o projeto.
- **Construção:** como o próprio nome já sugere esta fase refere-se ao momento em que o software é construído e preparado para implantação. Além da codificação em si, são criados os casos de teste e a documentação do sistema. Ao fim desta etapa já teremos um software em funcionamento.
- **Transição:** envolve os últimos estágios e agrega as últimas tarefas da atividade genérica de construção e marca a primeira etapa da implantação. O software é passado aos clientes e é realizado o treinamento e ajustes. Caso os objetivos do

ciclo de vida deste software precisem ser revistos com o passar do tempo basta iniciar outro ciclo de desenvolvimento.

Segundo Pressman (2010), ao mesmo tempo em que as etapas de construção, transição e produção estejam sendo conduzidas, o trabalho já esteja acontecendo em algum incremento de software, portanto, as fases do RUP não precisam ocorrem obrigatoriamente em sequência, mas em concorrência.

O RUP pode trabalhar de duas maneiras: de uma forma iterativa (iterações) ou de forma estática (workflows).

"Uma iteração é um ciclo completo de desenvolvimento, resultando em uma versão de um produto executável, que futuramente será somando ao produto final, e cresce de maneira incremental de uma iteração para outra." (RABELLO E BORTOLI, 2006, p.42).

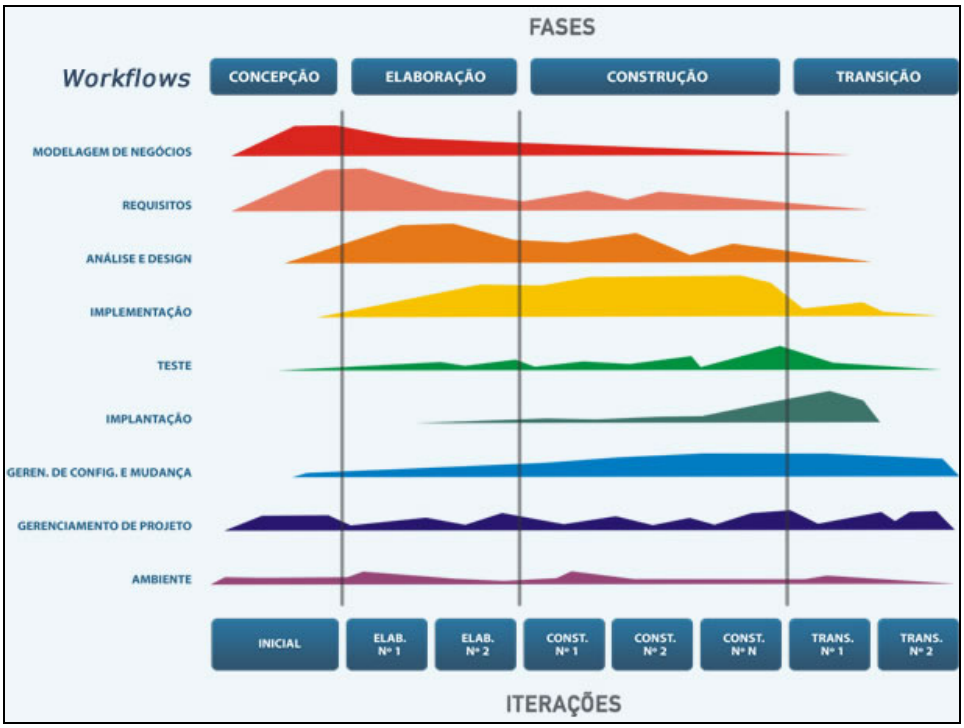
Os principais fluxos de trabalho do RUP, segundo Sommerville (2011), são os seguintes:

- **Modelagem do negócio:** descreve a estrutura e dinâmica do negócio por meio de casos de uso de negócios;
- **Requisitos:** descreve o método baseado em casos de uso para especificar os requisitos;
- **Análise e Projeto:** um modelo do projeto é criado e documentado com modelos de arquitetura e suas várias visões;
- **Implementação:** constitui o desenvolvimento do software com subsistemas, o teste da unidade e a integração;
- **Teste:** implica os casos de teste, procedimentos, e medidas para acompanhamento dos erros;
- **Implantação ou Entrega:** um *release* do produto é criado e passado aos usuários;
- **Gerenciamento da configuração:** constitui o apoio s gerencia das mudanças do sistema e a integridade dos elementos do projeto;
- **Gerenciamento do projeto:** gerencia o desenvolvimento como um todo, propondo estratégias de trabalho com um processo interativo;
- **Ambiente:** abrange a infra-estrutura necessária pra o desenvolvimento do software.

Segundo Rabello e Bortoli (2006), com o RUP os custos e os prazos são definidos na fase inicial do projeto, sendo que estas são fatores mais críticos do desenvolvimento de sistemas, dessa forma, os riscos do projeto são logo identificados pelo desenvolvedor, o que torna o processo flexível, permitindo e reavaliações e readequações.

O RUP é indicado para grandes projetos, mas pode suprir necessidades de pequenas equipes até grandes empresas de desenvolvimento, desde que exista envolvimento e disciplina entre os envolvidos no projeto. A gestão de projetos recomendada pelo RUP é disciplinada, envolve tarefas e responsabilidades dentro de uma organização de desenvolvimento de software.

Figura 10 - Modelo Básico do RUP.



Fonte: Repositório digital de imagens da Google (2013).

2.3.2 Modelos Ágeis

Esta sessão tem o objetivo de apresentar os métodos ágeis de desenvolvimento de software. Através dela será possível compreender a lógica destes modelos, o manifesto ágil, e

poderá identificar as diferenças entre desenvolvimento ágil e desenvolvimento dirigido a planos.

Entre os modelos de processo considerados métodos ágeis, que serão abordados têm-se:

- XP;
- DSDM (Método de Desenvolvimento Dinâmico de Sistemas);
- SCRUM;
- FDD (Desenvolvimento Guiado por Características).

2.3.2.1 XP (Extreme Programming)

A Programação Extrema (*Extreme Programming*), ou XP é um modelo de processos ágil criado por Kent Beck, em 1997, para um projeto para a Chrysler (fabricante de veículos norte-americana). É uma metodologia considerada uma evolução do modelo incremental, voltado para pequenas e médias empresas, e para projetos onde os requisitos não são bem definidos e mudam constantemente.

Segundo Wildt e Lacerda (2011), o XP tem como principal tarefa a codificação, com ênfase menor nos processos formais de desenvolvimento, com maior disciplina na codificação e testes.

“O XP é um conjunto de regras, valores e princípios que visam permitir o desenvolvimento rápido e eficiente de software, absorvendo a retroalimentação (*feedback*) relativa a mudança dos requisitos.” (RABELLO E BORTOLI, 2006, p.61).

É um modelo de processos que presa pela simplicidade, quanto mais simples for a solução para determinado problema melhor. Porém, a idéia de simplificar as resoluções de problemas, não significa tornar mais fácil, mas significa tornar um código que já esta funcionando de uma maneira, em algo ainda menos complexo, que continue sanando determinada necessidade.

Deve-se destacar também a flexibilidade do XP, pois é um modelo totalmente interativo, que permite mudanças constantes durante o desenvolvimento, porém isso exige grande comprometimento por parte dos membros da equipe.

Segundo Rabello e Bortoli (2006), os desenvolvedores, gerentes e os clientes são considerados membros da equipe que compõem o projeto, uma vez que esta metodologia tem grande ênfase no trabalho em equipe.

A XP é um legado de outros métodos já utilizados, desta forma, incorpora algumas características destes modelos, por exemplo, os requisitos são expressos por “histórias do usuário”, que vem a ser casos de usos que especificam os requisitos e, posteriormente são implementados diretamente como uma série de tarefas.

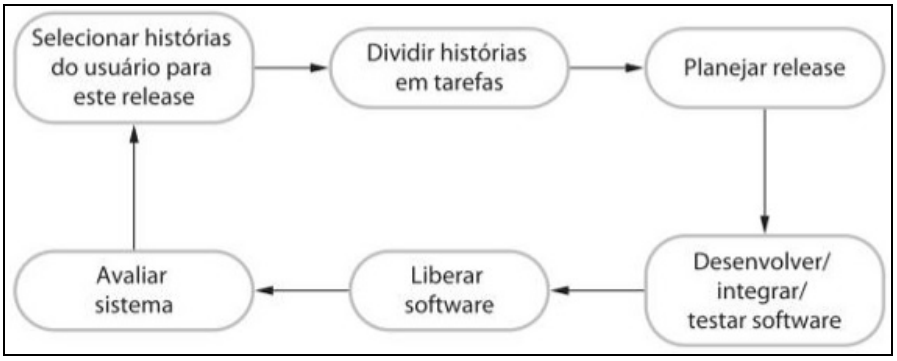
O modelo XP requer muito envolvimento do cliente para especificação de requisitos e priorização destes requisitos, para definição do direcionamento das tarefas.

Desta forma, segundo Sommerville (2011), o cliente do sistema faz parte da equipe de desenvolvimento e discute os cenários com os outros membros.

A organização do projeto em XP é baseada em iterações (*releases*), que é onde se encontram a programação, testes e integração. Sendo que, a cada iteração podem surgir novas estórias e requisitos, e com isso, teremos novas tarefas, que serão incorporados ao projeto nos releases posteriores (Figura 11).

Conforme Rabello e Bortoli (2006), à medida que as iterações vão sendo realizadas, vão sendo feitos testes de aceitação, que após a aprovação é passado para o cliente, lembrando que os testes são realizados, tato por desenvolvedores, quanto pelos clientes o que facilita a identificação de problemas ou de novas necessidades.

Figura 11 - Ciclo de um release em XP.



Fonte: Sommerville (2011, p.44).

Além de tudo isso, segundo Campelo (2003), a XP tem como base quatro valores, considerados como guias para as equipes que utilizam a metodologia, são eles:

- **Comunicação:** parte dos problemas que acontecem em alguns projetos são atribuídos a falta de comunicação, portanto, o XP presa por este fator, empregando práticas que estimulem a comunicação entre a equipe e os clientes, forçando os membros ao dialogo oral, como um exemplo destas práticas, temos a programação em pares utilizada no XP;
- **Simplicidade:** procura reduzir ao máximo a complexidade através de soluções simples. Procura a solução mais simples possível, mas que possa funcionar.
- **Feedback:** significa obter a opinião do cliente sobre o que foi feito o mais rápido possível, a fim de reduzir os falsos requisitos. O *feedback* do cliente é estimulado através de *releases* curtos e entrega contínua de software de valor. O andamento do projeto é reportado a todos através de reuniões diárias e métricas simples, coletadas e reportadas com frequência;
- **Coragem:** é necessária a presença de coragem para poder realizar os valores citados anteriormente, pois, é preciso coragem para simplificar um código que já esta funcionando para deixá-lo mais simples, coragem para fazer alterações constantes no processo de desenvolvimento e para expor ao cliente a realidade do processo implementado.

Conforme Rabello e Bortoli (2006), além destes valores, a programação extrema possui várias práticas que formam a base do desenvolvimento (Tabela 2).

Tabela 2 – Práticas da XP.

Prática	Descrição
Logo de planejamento	O Cliente elabora um plano para o projeto, avalia as funcionalidades que devem ser implementadas e estabelece prioridades; as estórias são estimadas para que conheçam os custos. Em cada iteração deve implementado somente o necessário.
Stand up meetting	Reunião diária para avaliar o trabalho do dia anterior e planejar o dia que se inicia. Tem duração de quinze minutos e deve ser realizada em pé por todos os membros da equipe de desenvolvimento.
Realeses curtos (pequenas iterações)	O software será entregue em pequenas versões, contendo as funcionalidades expostas pelo cliente na iteração anterior.
Metáfora	Tem poder de transmitir idéias complexas de forma simples, estabelece uma linguagem comum entre os membros da equipe de desenvolvimento e o cliente.
Design (projeto) simples	Deve desenvolvido realmente o necessário para iteração atual e projetada “a coisa mais simples que possa funcionar”.

Desenvolvimento guiado por testes	Os testes são realizados pelos desenvolvedores e pelo cliente. Os desenvolvedores escrevem testes unitários executados com frequência durante o desenvolvimento. Enquanto o cliente realiza testes de aceitação sobre o que foi definido para a iteração.
Refatoramento (*refactoring*)	Com a finalidade de facilitar a implementação das mudanças é necessário refatorar o código, que significa seu melhoramento sem alterar sua funcionalidade.
Programação em par	Há mais de um desenvolvedor utilizando a mesma máquina. Dessa forma, o código é revisado por mais de um programador o que favorece o aprendizado, a comunicação, a diversidade de idéias e a economia de tempo. Mas é necessário definir um padrão de codificação.
Código coletivo	Toda equipe é responsável pelo código. Sendo assim, não é necessário pedir permissão para alterá-lo. São criados mecanismos de revisão e verificação de código.
Integração continua	Diversos módulos são integrados várias vezes ao dia e os testes devem ser realizados em cada iteração para evitar problemas.
Ritmo sustentável (semana de 40 horas)	Essa prática defende que trabalhar por períodos muito extensos não é produtivo. Um fator importante para a produtividade do projeto é o bem estar da equipe, desta forma, aumenta também a qualidade do trabalho.
Cliente presente	O cliente faz parte da equipe e divide o mesmo ambiente.
Código padronizado	Todo código-fonte escrito é formatado de acordo com os padrões.

Fonte: Rabello e Bortoli (2006, p.64).

O XP é um modelo muito útil e fácil de ser implantado em qualquer empresa de desenvolvimento, e de qualquer tamanho, que necessite de um processo com rápido desempenho e simplicidade. Porém, apesar de todos os benefícios a metodologia pode apresentar problemas quanto a evolução do software, pois o processo é focado nas pessoas e não na codificação, por esse motivo, temos pouca documentação gerada no decorrer do processo, o que pode vir a ocasionar estes imprevistos.

2.3.2.2 DSDM (Dynamic Systems Development Method)

Segundo Rabello e Bortoli (2006), o *Dynamic Systems Development Method* (DSDM), ou Método de Desenvolvimento de Sistemas Dinâmicos é uma metodologia ágil que surgiu em 1994, baseado no *Rapid Aplication Development* (RAD), que visa criar em manter sistemas que mantém restrições de prazo apertadas, desta forma, é focado em tempo e recursos e não na definição de funcionalidades.

Segundo Rabello e Bortoli (2006), o DSDM se baseia em nove princípios básicos que incorporam e refletem a essência do desenvolvimento ágil, sendo eles:

- Usuário ativo;
- As equipes devem ter autonomia na tomada de decisões;
- Foco na entrega frequente de produtos;
- Adaptação às tarefas requisitadas;
- Desenvolvimento interativo e incremental;
- As mudanças durante o desenvolvimento são reversíveis;
- Os requisitos são descritos em alto nível, sem muito detalhamento ou documentação;
- Os testes são parte do ciclo de vida;
- A colaboração de todos os envolvidos no projeto é primordial.

Conforme Pressman (2010) o DSDM possui seu ciclo de vida definido por três ciclos interativos diferentes, precedidos por duas atividades adicionais:

- **Estudo da viabilidade:** são definidos os requisitos básicos e as restrições do projeto e depois é avaliada a viabilidade da utilização do DSDM na aplicação em contexto;
- **Estudo do negócio:** estabelece os requisitos e a informação necessária para aplicação e para fornecer valor ao negócio, além de definir a arquitetura básica da aplicação;
- **Iteração do modelo funcional:** são criados um grupo de protótipos funcionais para demonstrar a funcionalidade ao cliente, ressaltando que todos os protótipos são gerados para evoluir para o software final. Esta etapa tem o objetivo de coletar mais requisitos para o sistema através do *feedback* do cliente sobre os protótipos.
- **Iteração de projeto e construção:** revisa os protótipos da etapa anterior, buscando validá-los, para que possam agregar valor ao produto final. Em alguns casos a iteração do modelo funcional acontece simultaneamente a interação de projeto e construção.

- **Implementação:** é a etapa onde o ultimo incremento é colocado em operação. Fica em aberta então a agregação de novas modificações ao software. O processo de desenvolvimento do DSDM contínua por meio do retorno a atividade de iteração do modelo funcional.

Para poder utilizar esta metodologia a empresa precisa ter o processo com o mínimo de organização, pelo fato de ele focar em um dos pontos mais críticos no desenvolvimento, o tempo.

Quanto à utilização do DSDM, Rabello e Bortoli (2006), afirmam que ele pode ser usado em projetos de qualquer dimensão, porém, em projetos maiores é aconselhável dividir as etapas em pequenas partes para que a execução das tarefas possa seguir seus princípios com maior precisão.

"O DSDM pode ser combinado com o XP para fornecer uma abordagem combinada que defina um modelo de processo sólido (o ciclo de vida DSDM) com práticas instrumentais (XP) necessárias a construção de incrementos de software." (PRESSMAN, 2010, p.69).

O método DSDM pertence a um consócio, o DSDM *Consortium*, que é um grupo mundial de empresas que zela pelo modelo de processos. Para ter acesso a documentação mais detalhada sobre a metodologia é necessário cadastramento e pagamento de taxas mensais a este consórcio, por isso o DSDM é pouco conhecido.

2.3.2.3 SCRUM

Segundo Rabello e Bortoli (2006), o Scrum é uma metodologia de desenvolvimento ágil que foi criada na década de 1990 pelas empresas *Advanced Development Methods* e VMARK, por Ken Schawber e Jeff Sutherland. O foco principal do Scrum esta no gerenciamento de projetos.

Muitos definem o Scrum não como uma técnica ou um processo, mas como um *framework* onde você pode agrupar diversas práticas, processo de técnicas diferentes.

"O papel do Scrum é fazer transparecer a eficácia relativa das suas práticas de desenvolvimento para que você possa melhorá-las, enquanto provê um *framework* dentro do qual produtos complexos podem ser desenvolvidos." (SCHWABER, 2009, p.3).

Esta metodologia se baseia em alguns princípios do XP, onde encontramos equipes pequenas, requisitos pouco conhecidos ou pouco estáveis e iterações curtas.

A diferença entre o direcionamento do XP e do Scrum é que a primeira tem o foco na programação, enquanto a outra esta voltada para o gerenciamento do projeto sem definir técnicas ou ferramentas, apenas define como serão realizadas as tarefas.

Segundo BISSI (2007), a metodologia Scrum apenas estabelece conjuntos de regras e práticas de gestão que devem ser adotadas para garantir o sucesso de um projeto e é centrado, primordialmente, no trabalho em equipe, melhora a comunicação e maximiza a cooperação, permitindo que cada um faça o seu melhor e se sinta bem com o que faz o que mais tarde se reflete num aumento de produtividade.

Existem basicamente dois tipos de processos dentro do gerenciamento de projetos de software: processos definidos e processos empíricos.

Segundo Rabello e Bortoli (2006), os processos definidos são considerados simples e reprodutíveis, são exemplos deste o modelo Cascata e o Espiral. Enquanto os empíricos são processos não predefinidos, sendo utilizados quando encontramos atividades mais complicadas, onde o andamento não pode ser previsto e nem replicado. O Scrum se baseia em processos empíricos.

Ainda quanto aos objetivos da metodologia, Rabello e Bortoli (2006), afirmam que o Scrum busca entregar o software com a maior qualidade possível dentro de ciclos ou séries compostas por outros pequenos espaços de tempo chamados de *Sprints*. Cada *Sprint* tem, aproximadamente, um mês de duração.

Quanto à organização das tarefas Pressman (2010, p.69) diz que:

> Os princípios do Scrum são usados para guiar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades de arcabouço: requisitos, análise, projeto, evolução e entrega. Em cada atividade de arcabouço, as tarefas são organizadas e adaptadas dentro dos *Sprints*.

Conforme Schwaber (2009), o método possui três pilares de sustentação, sendo eles:

- **Transparência:** garante que aspectos do processo que afetam os resultados fiquem visíveis a todos que gerenciam e participam do projeto;
- **Inspeção:** os aspectos do processo devem ser inspecionados constantemente para que variações inaceitáveis sejam detectadas;
- **Adaptação:** se o inspetor determinar, a partir da inspeção, que um ou mais aspectos do processo estão fora dos limites aceitáveis e que o produto resultante será inaceitável, ele deverá ajustar o processo ou o material sendo

processado. Esse ajuste deve ser feito o mais rápido possível para minimizar desvios posteriores.

A metodologia Scrum tem o foco no gerenciamento do trabalho, ou seja, na organização das tarefas, na programação e na máxima busca qualidade do produto final. São definidas as tarefas para cada *sprint* e ao final de cada uma, procura entregar uma parte do software, visando obter melhores respostas e resultados.

O vocabulário utilizado na metodologia tem relação com seu conjunto de regras e práticas gerenciais, que conforme Rabello e Bortoli (2006, p.70), são detalhadas da seguinte maneira:

- **Scrum Master:** é quem garante a utilização das práticas, regras e valores do Scrum por parte de toda a equipe. É o responsável pela gerencia do projeto e pela aplicação das regras, sendo assim, o representante da equipe, geralmente, quem se ocupa desta tarefa são engenheiros de software.
- **Product Owner:** representa a voz do cliente e é responsável por garantir que a equipe agregue valor ao negócio. O *Product Owner* escreve centrado nos itens do cliente (histórias tipicamente do usuário), os prioriza e os adiciona para o *product backlog*. Equipes de Scrum devem ter um *product owner*, e, embora esse possa também ser um membro da equipe de desenvolvimento, recomenda-se que este papel não seja combinado com o de *Scrum Master*.
- ***Backlog* do Produto:** consiste em uma lista onde estarão todas as funcionalidades desejadas, requisitos, funções e tecnologias empregadas, além das tarefas necessárias para construir o software e atingir os objetivos do projeto. As tarefas são organizadas por ordem de prioridades.
- **Sprint:** ciclo de desenvolvimento do Scrum, que tem, aproximadamente, trinta dias. Neste período, as tarefas são realizadas, após terem sido divididas conforme as prioridades, conforme citado anteriormente, procurando ao fim do ciclo entregar uma parte do software ou incremento funcional ao cliente.
- **Equipe Scrum:** fazem parte da equipe Scrum tanto desenvolvedores, quanto os clientes. Temos equipes pequenas de no máximo sete pessoas. Recomenda-se que a equipe seja auto-organizada e auto conduzida.

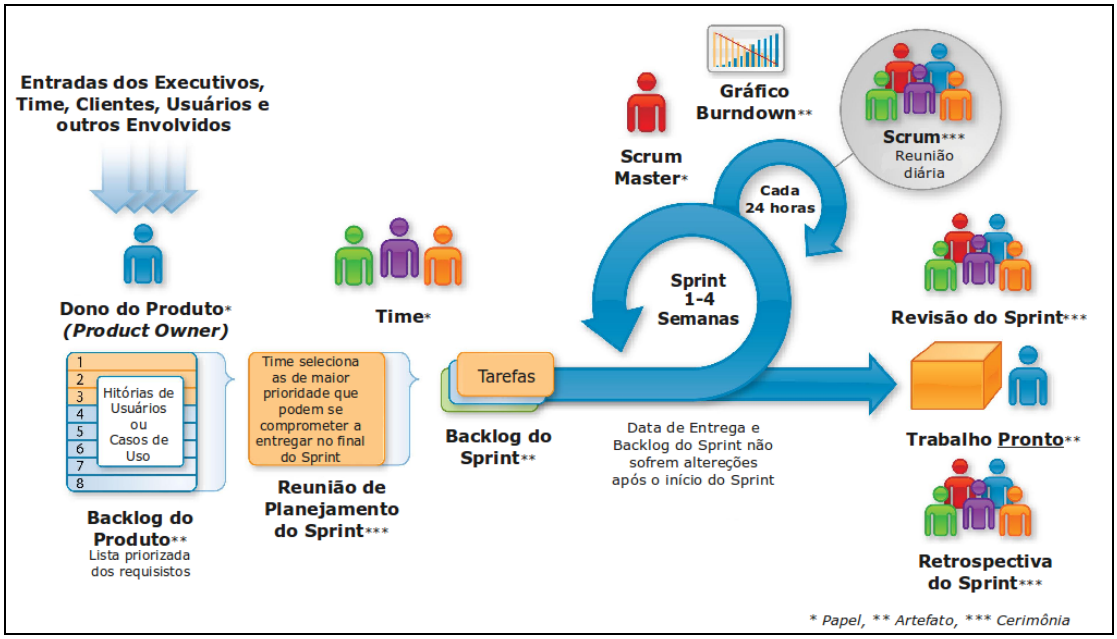
- **Sprint Backlog:** é a lista de tarefas ou funcionalidades que devem ser desenvolvidas durante uma determinada *sprint*.
- **Reuniões de Scrum diárias:** a equipe se reúne diariamente para discutir durante quinze minutos sobre o que cada componente conseguiu finalizar deste a última reunião, as dificuldades e o que pretendem executar até a próxima reunião, as reuniões diárias devem ser realizadas com todos os participantes em pé.
- **Reunião de Planejamento da Sprint:** envolve todos os interessados no projeto, mas geralmente basta apenas ter a presença da equipe de desenvolvimento, para discutir e definir os itens da lista de *backlog* do produto com a finalidade de organizar as tarefas e definir como o trabalho será conduzido durante a *sprint*, além da definição das prioridades. Esta reunião deve ter a duração de quatro horas.
- **Revisão da Sprint:** a revisão da *sprint* é executada no final da *sprint* para inspecionar o incremento e adaptar o *backlog* do produto se necessário. É uma reunião informal, e a apresentação do incremento destina-se a motivar e obter comentários e promover a colaboração. O resultado da reunião de revisão da *sprint* é um *backlog* do produto revisado que define o provável *backlog* do produto para a próxima *sprint*. Deve ter a duração de quatro horas para uma *sprint* de um mês.
- **Retrospectiva da Sprint:** a retrospectiva da *sprint* é uma oportunidade para o a equipe de desenvolvimento inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima *sprint*. Esta reunião deve acontecer após a revisão e antes da reunião de planejamento da próxima *sprint* e deve ter duração de três horas para uma *sprint* de um mês. Representa a busca pela melhoria continua do processo de desenvolvimento.
- **Gráfico Burndown:** é o gráfico que representa o andamento das atividades em uma *sprint*, através da comparação do andamento planejado pela equipe, com o andamento real das atividades.

Com todas estas práticas e regras trabalhando unidas, temos uma metodologia de desenvolvimento que proporciona, quando seguida com disciplina por todos os envolvidos, total controle sobre o andamento do projeto, sobre as necessidades do cliente e que permite

com o passar do tempo avaliar e medir a velocidade e capacidade da equipe de desenvolvimento.

Segundo Rabello e Bortoli (2006), o ciclo Scrum é novamente iniciado após a entrega do incremento ao cliente (Figura 12). Enquanto tiverem funcionalidades listadas no *backlog* do produto a serem agregadas ao sistema, estarão sendo criados novos ciclos de desenvolvimento.

Figura 12 - Fluxo do processo Scrum.



Fonte: Repositório digital de imagens da Google (2013).

O Scrum é uma metodologia ágil que pode trabalhar tranquilamente com projetos mais complexos, onde os requisitos não são claros e mudam com frequência, desta forma, em alguns casos de projetos menores é difícil aplicar o Scrum pelo fato de exigir bastante disciplina dos envolvidos, no sentido de que se a equipe pular uma etapa do ciclo de desenvolvimento ou deixar de fazer uma determinada reunião, por exemplo, o processo já não será caracterizado como Scrum.

Para aplicar qualquer modelo de processos é necessário analisar a situação em contexto para que seja feito o emprego do mais viável.

2.3.2.4 FDD (Feature Driven Development)

O *Feature Driven Development* (Desenvolvimento guiado por Características), ou simplesmente FDD, é modelo de processos, criado por Peter Coad e Jeff De Luca, durante um grande projeto implementado por eles no Singapura, onde o desenvolvimento é guiado por características (*features*).

Segundo Pressman (2010), no contexto do FDD, uma característica é uma função valorizada pelo cliente que pode ser implementada em duas semanas ou menos.

Desta forma, temos o FDD como um método que se organiza com ciclos de desenvolvimento curtos, onde as características apontadas pelo cliente podem ser desenvolvidas, como citado anteriormente, ciclos de, no máximo, duas semanas.

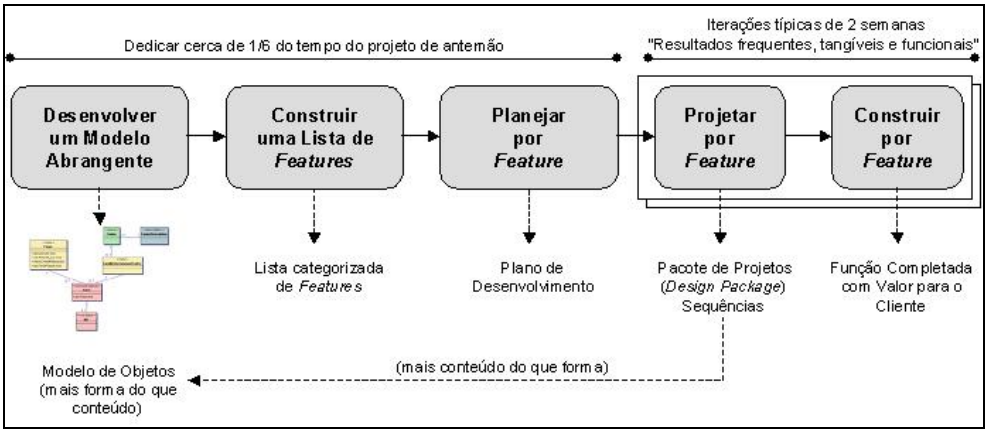
Segundo Rabello e Bortoli (2006), se houver necessidade de o ciclo ser maior, as características devem ser divididas em características menores, para que possam se adequar a esta restrição do FDD quanto ao tempo de duração de seus ciclos.

O fato de as características serem divididas em pequenos blocos de funcionalidade passíveis de entrega constante, os usuários encontram maior facilidade para descrever o sistema e as representações de projeto e de código são mais fáceis de serem inspecionadas.

"Este modelo concentra-se em cinco principais etapas: construir um modelo inicial, identificar, planejar e classificar as características e interativamente fazer do projeto e da construção de cada característica seu foco principal." (HARTMANN *apud* RABELLO E BORTOLI, 2006, p.73).

Estas cinco principais etapas caracterizam o ciclo e os processos de desenvolvimento do FDD (Figura 13).

Figura 13 - Processos do FDD.



Fonte: Repositório digital de imagens da Google (2013).

O projeto tem inicio com a criação do modelo abrangente para obter-se uma visão geral do projeto. Na sequência é feita uma lista de todas as características necessárias, sendo que estas são todas as funcionalidades do projeto. O processo seguinte é o planejamento, a partir das funcionalidades e requisitos encontrados e listados anteriormente. No planejamento serão definidas as prioridades de características, o tempo de desenvolvimento para cada uma e de que maneira serão atingidos os objetivos. Feito isso, escolhe-se um conjunto de características a serem projetadas e implementadas.

Segundo Pressman (2010), o FDD é o que coloca mais ênfase em diretrizes e técnicas de gestão de projeto entre os métodos ágeis e é essencial que os desenvolvedores, seus gerentes e o cliente, enfim todos os envolvidos no projeto, entendam o estado do projeto, que avanços foram feitos e os problemas encontrados.

A metodologia guiada por características apresenta uma grande vantagem sobre os riscos do projeto, que são menores pelo fato de serem produzidos como resultado do formato de constante entrega do produto em pleno funcionamento.

Segundo Rabello e Bortoli (2006), o FDD facilita a comunicação e a medição do progresso do processo de acordo com a lista de características, possuindo um desenvolvimento interativo que fornece visibilidade sobre o projeto.

Pode-se considerar que não seja um método completo e robusto, porém, apresenta todos os quesitos para construção de um software. Sendo assim, conforme Rabello e Bortoli (2006), um modelo de processos adequado para projetos de tamanho médio, onde os requisitos mudam constantemente.

2.4 EVOLUÇÃO DO SOFTWARE

O desenvolvimento de um software não chega ao fim quando ele é entregue ao cliente, mas, independentemente de seu tamanho e complexidade, este vai evoluir com o passar do tempo, e as mudanças serão constantes e inevitáveis por toda sua vida útil.

Segundo Araújo, Souza e Vale (2011), os sistemas sempre procuram reproduzir situações do mundo real, desta forma, existe a necessidade de que o software mude acompanhando as mudanças de requisitos do ambiente em que está inserido.

Quando o software não sofre essas mudanças acaba ficando obsoleto, perdendo seu valor, uma vez que não atende aos seus objetivos e conseqüentemente pode vir a cair em desuso.

Conforme Sommerville (2011), depois que o sistema é implantado, para que ele se mantenha útil é imprescindível que ocorram as mudanças, que podem ser mudanças nos negócios, mudanças nas expectativas dos usuários, correção de erros, atualização do sistema, aperfeiçoamento do software ou adaptação a uma nova realidade. Sendo que, a evolução de software compreende as mudanças que irão ocorrer em um programa a fim de deixá-lo completo e, se possível, livre de erros.

A evolução de software estabelece processos de melhoria de um software já desenvolvido, ou que está sendo desenvolvido, visando prever e corrigir quaisquer problemas que possam ocorrer, ou que venham a ocorrer, durante a utilização de um produto de software.

“A maioria das grandes empresas gasta mais na manutenção de sistemas existentes do que no desenvolvimento de novos sistemas.” (SOMMERVILLE, 2011, p.164).

As empresas investem grandes quantias no desenvolvimento de seus softwares e conforme vão alimentando suas bases, tornam-se dependentes destes sistemas. Tendo assim, os sistemas como participantes ativos em seus negócios, precisando investir na sua evolução para continuar agregando valor aos negócios da empresa.

“Aproximadamente, 85% a 90% dos custos organizacionais de software são custos de evolução, o que ressalta ainda mais a importância deste processo de melhoria continua na vida de um software.” (ERLIKH *apud* SOMMERVILLE, 2011, p.164).

Conforme Sommerville (2011), sistemas úteis geralmente tem uma vida útil extensa, por exemplo, sistemas militares ou de infra-estrutura, como um software de controle do tráfego aéreo podem ter uma vida útil de aproximadamente uns 30 anos. Sendo assim, mais uma causa pela qual as empresas investem tanto na manutenção de seus sistemas.

Segundo Nascimento (2010), sabe-se que o processo de envelhecimento de um software é inevitável, mas que pode ser previsto para podermos tomar medidas que limitem seus efeitos. Sendo assim, existem dois tipos de envelhecimento: o primeiro ocorre quando os responsáveis por um software falham em adaptá-lo aos novos requisitos; e o segundo ocorre devido ao resultado provocado pela forma como as mudanças são realizadas no software, mudanças essas que tinham o objetivo de satisfazer esses requisitos que mudaram.

O envelhecimento de um software pode trazer vários problemas, como por exemplo, a perda de desempenho do sistema caso as alterações previstas não tenham sido feitas da maneira correta, podendo acarretar novos erros com o passar de sua utilização.

Para julgar se um software necessita de evolução, ou precisa ser substituído, uma série de fatores devem ser levados em consideração.

Para Araújo, Souza e Vale (2011), entre os fatores que devem ser levados em conta na evolução ou construção de um novo sistema temos:

- Custo envolvido no processo;
- A confiabilidade do software após a manutenção;
- A capacidade de se adaptar a mudanças futuras;
- Desempenho;
- Limitações das atuais funcionalidades;
- Tendências de mercado que atendam o problema de alguma maneira mais viável.

2.4.1 PROCESSOS DE EVOLUÇÃO

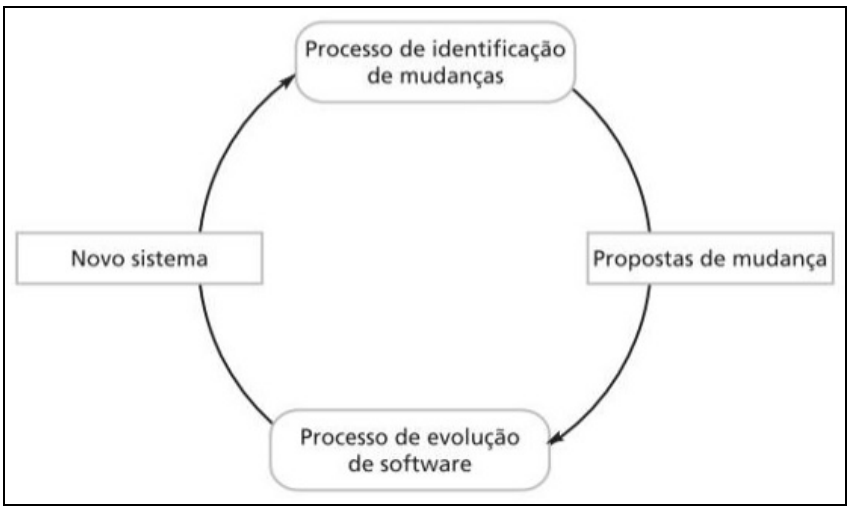
Os processos de evolução de um software podem apresentar-se de diversas maneiras, dependendo do ambiente que gira em torno do software. Isso acontece, pois em qualquer empresa ou organização uma proposta de mudança funciona como um acionador da evolução de seus softwares.

Segundo Sommerville (2011), o tipo de software que esta sendo mantido, os processos de desenvolvimento utilizados em uma organização e as habilidades das pessoas envolvidas no projeto, são alguns fatores que podem fazer a evolução dos processos de software variar de um caso para outro.

Desta forma, algumas empresas tratam a evolução de software como um processo informal e as solicitações de mudanças, geralmente, são produto da interação e conversas diretas entre usuários do sistema e desenvolvedores. Enquanto isto, em outras empresas, tratam a evolução de software como um processo formal todo documentado e bem estruturado.

“Os processos de identificação de mudanças e de evolução de sistemas são cíclicos e continuam durante toda a vida de um sistema (Figura 14).” (SOMMERVILLE, 2011, p.166).

Figura 14 - Processos de identificação de mudanças e de evolução.



Fonte: Sommerville (2011, p.166).

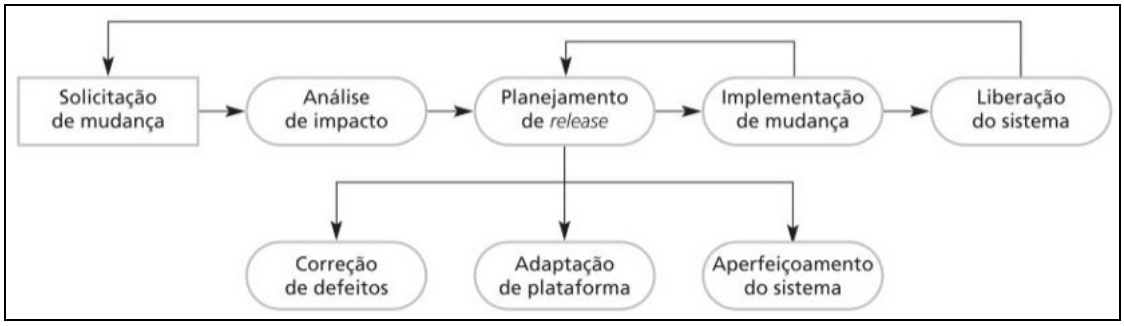
Como pode-se notar, o processo cíclico de evolução de um software tem inicio com o processo de identificação das mudanças, que podem vir de solicitações de novas funcionalidades e requisitos no software, da identificação de falhas ou através de propostas de melhorias por parte da equipe desenvolvimento.

Após serem identificadas as mudanças necessárias são criadas as propostas de mudanças, em seguida são realizadas as alterações que vão resultar em uma nova *release* ou nova versão do sistema.

Para Araújo, Souza e Vale (2011), deve-se sempre avaliar a documentação e códigos existentes, juntamente com sua arquitetura, estrutura de dados e interface, para que seja possível identificar modificações necessárias, avaliar custos e os impactos destas alterações.

Sendo assim, toda solicitação de mudança deve ser avaliada, planejada e implementada, e o processo se repete a cada nova solicitação (Figura 15).

Figura 15 - Processo de evolução de software.



Fonte: Sommerville (2011, p.167).

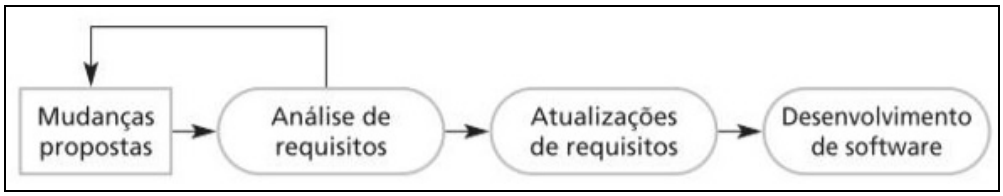
Como se pode notar, a Figura 16 traz uma visão geral do processo de evolução com todas as suas atividades fundamentais.

Segundo Sommerville (2011), o processo inicia com a solicitação da mudança, então são incluídas as atividades de avaliação e análise do custo e do impacto, para ter noção de quanto o sistema em funcionamento será afetado e quanto irá custar para implementação das mudanças. Se as mudanças forem viáveis, um novo *release* será planejado, considerando correção de defeitos, adaptação de plataforma e aperfeiçoamento do sistema, como possibilidades de mudanças. Feito isso as mudanças são implementadas e validadas, e uma nova versão do sistema é liberada para uso.

Conforme surgem as necessidade de novas alterações o processo de evolução se inicia novamente, passando por todas as etapas apresentadas.

Para Sommerville (2011), a implementação da mudança deve atualizar a documentação que corresponde ao processo de evolução do software para refletir as alterações do sistema (Figura 16), podendo ser documentos de especificação, modelos de projeto, implementação, testes, entre outros.

Figura 16 - Implementação da Mudança.



Fonte: Sommerville (2011, p.167).

Araújo, Souza e Vale (2011) destacam alguns problemas e alguns efeitos colaterais causados por estes problemas, que podem ocorrer durante o processo de manutenção, sendo os principais problemas:

- Ausência ou deficiência na documentação;
- Dificuldade na identificação de alterações realizadas anteriormente;
- Falta de controle das alterações e de versões;
- Baixa manutenibilidade do software.

Entre os principais efeitos colaterais que ocorrem com a realização destes processos podemos destacar os seguintes:

- Modificação da estrutura do código;
- Inclusão de códigos com erros;
- Desatualização da documentação.

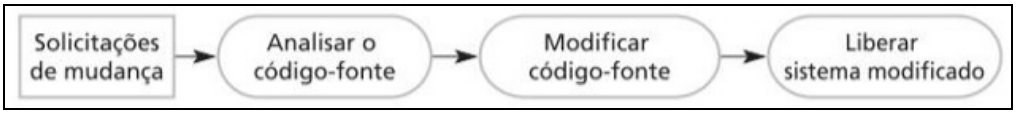
Como podemos ver o processo de evolução e de implementação de mudanças se assemelha em algumas atividades ao processo de desenvolvimento inicial do sistema, uma vez que é necessário entender os requisitos e a regra de negócio do programa, para que sejam evitados os problemas descritos anteriormente.

Compreender o programa significa entender como a funcionalidade a ser alterada encontra-se implementada e avaliar o impacto da mudança em outras partes do software e no ambiente onde ele se encontra.

Segundo Sommerville (2011), existem casos em que a mudança deve ser realizada rapidamente, o que significa que não pode acompanhar o processo de análise formal da mudança.

Desta forma, estes tipos de mudanças, conhecidas como correções de emergência, tem prioridade sobre a documentação, e em vez de se modificar requisitos e o projeto e serem seguidas as atividades normais de desenvolvimento, são realizadas alterações diretas no código, com o objetivo de resolver o problema o mais rápido o possível (Figura 17).

Figura 17 - Processo de Correção de Emergência.



Fonte: Sommerville (2011, p.168).

Conforme Araújo, Souza e Vale (2011), algumas solicitações de mudanças urgentes podem ocorrer nos seguintes casos, por exemplo:

- Se ocorrer um defeito grave que impede o funcionamento normal do sistema;
- Alterações do ambiente impedem o funcionamento do sistema;
- Alteração no negócio do cliente ou uma nova legislação;
- Entrada de um novo concorrente no mercado.

Contudo, temos as correções emergenciais como a solução mais rápida e viável para as situações citadas anteriormente, porém, não pode ser considerada a melhor solução.

Segundo Sommerville (2011), o uso de correções emergenciais acelera o envelhecimento do software, uma vez que elas diminuem a sua manutenibilidade, tornando as alterações mais difíceis e os custos com evolução mais altos.

2.4.2 DINÂMICA DA EVOLUÇAO

A dinâmica da evolução se trata do estudo das mudanças nos sistemas de informação e baseia-se nas oito leis de Lehman.

“Nas décadas de 1970 e 1980, Lehman e Belady (1985) realizaram vários estudos empíricos sobre a mudança de sistema com intenção de compreender mais sobre as características de evolução de software.” (SOMMERVILLE, 2011, p.169).

A evolução e envelhecimento de vários sistemas foram examinados com o objetivo de criar uma teoria padronizada sobre os processos de evolução de software. A partir destes estudos surgiram as oito Leis de Lehman.

Segundo Nascimento (2010), estas leis se aplicam a qualquer software que resolva um problema ou implemente uma solução computacional a alguma situação do mundo real. Sendo elas:

- **Mudança contínua:** um sistema de informação que é usado, deve ser constante e continuamente adaptado, caso contrário, se torna progressivamente menos satisfatório e perde valor de negócio, a medida que não atende a todas as expectativas e necessidades dos usuários.
- **Complexidade Crescente:** à medida que um programa é alterado, a sua complexidade cresce, a não ser que sejam tomadas ações para evitar ou diminuir o aumento da complexidade.
- **Auto-regulação:** o processo de evolução é auto-regulado, pois atributos do sistema como tamanho, tempo entre versões e números de erros reportados é geralmente é quase invariável.
- **Estabilidade organizacional:** ao longo de sua vida, a taxa de desenvolvimento de um software é constante e independente dos recursos destinados ao desenvolvimento do sistema.

- **Conservação da familiaridade:** durante a vida produtiva de um software em evolução, o índice de alterações e versões é constante e estatisticamente invariante.
- **Crescimento contínuo:** a funcionalidade oferecida pelos sistemas tende aumentar para manter a satisfação do usuário.
- **Qualidade decrescente:** a qualidade dos programas cairá, a não ser que eles sejam rigorosamente mantidos e adaptados às mudanças do ambiente em que estão inseridos.
- **Sistema de retorno (*feedback*):** os processos de evolução incorporam sistemas de retorno multiagentes, *multiloop*, e você deve tratá-los como tais para que sejam melhorados com sucesso.

Conforme Sommerville (2011), as observações de Lehman parecem sensatas, porém devem ser levadas em consideração ao se planejar o processo de manutenção.

2.4.3 MANUTENÇÃO DE SOFTWARE

"A manutenção de software é o processo geral de mudança em um sistema depois que ele é liberado para uso." (SOMMERVILLE, 2011, p.170).

A manutenção de software compõe as atividades que acontecem após a entrega de um software, com a finalidade de adaptar, implementar mudanças necessárias, corrigir defeitos e agregar valor a este software para que ele continue sendo útil para os usuários.

Este processo normalmente inicia-se por uma solicitação do cliente ou por algum relatório de problemas gerado pelo usuário.

Segundo Portella (2010), ainda que reparar defeitos seja a grande atividade da fase de manutenção, existem outras causas que acabam por gerar a necessidade do produto de software passar por uma manutenção, tais como:

- Mudança na legislação pertinente ao sistema;
- Mudança na tecnologia de TI/SI;
- Mudança no processo-alvo;
- Mudança no produto;
- Mudança na clientela;

- Mudança na direção;
- Mudança nos modelos de gestão etc.

Conforme Araújo, Souza e Vale (2011), as ações ligadas à atividade de manutenção de software foram classificadas de acordo com sua natureza em quatro categorias:

- **Manutenção Corretiva:** é a manutenção que trabalha com erros de funcionalidade emergenciais, em ocorrências de falhas e problemas, causadas por erros e defeitos durante a utilização do software.
- **Manutenção Adaptativa:** refere-se a manutenção que modifica um software para se adaptar a outro ambiente externo, como novas versões de sistemas operacionais, diferentes bancos de dados, entre outros elementos.
- **Manutenção Perfectiva:** significa a inclusão de novos requisitos, acréscimo de funcionalidades, modificações de funcionalidades em funcionamento e ampliações. Geralmente são mudanças realizadas a partir da solicitação dos usuários.
- **Manutenção Preventiva:** são realizadas alterações no software com a finalidade de aumentar a confiabilidade e manutenibilidade do sistema. Essas alterações devem ser baseadas sobre a avaliação continua das interfaces internas e externas, com o objetivo de evitar interrupções indesejadas.

Na prática, na maioria das vezes, não temos essa distinção tão clara dos tipos de manutenção, principalmente em empresas de pequeno e médio porte.

De acordo com Pressman (2010), a manutenção de software pode ser responsável por mais de 70% de todo o esforço despendido por uma organização. E essa porcentagem continua aumentando à medida que mais software é produzido.

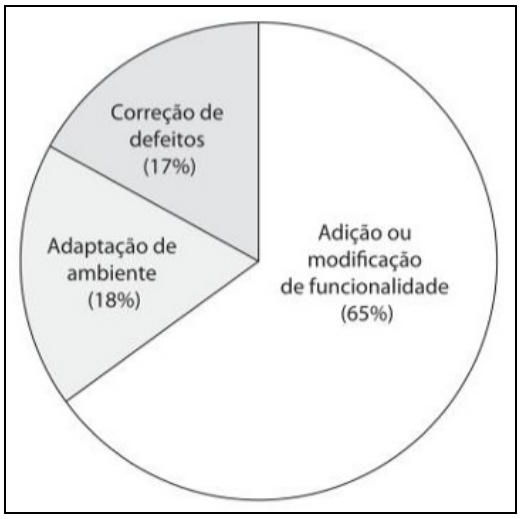
Segundo Sommerville (2011), que a maior parte dos gastos de manutenção concentra-se na adição de funcionalidades e na implementação de novos requisitos, desta forma, sendo superiores ao que é gasto na correção de defeitos.

“Um esforço de desenvolvimento que custe 25,00 por linha de código, pode custar 1000,00 por linha mantida.” (PORTELLA, 2010, p.8).

Apesar de todas estas evidências, os custos relativos a manutenção podem variar de acordo com o tipo de software, considerando o seu nível de complexidade e tamanho.

A Figura 18 apresenta, aproximadamente, a distribuição dos esforços e dos custos de manutenção.

Figura 18 - Distribuição dos esforços de manutenção.



Fonte: Sommerville (2011, p.171).

Conforme Sommerville (2011), os custos com manutenção de software continuam aumentando com o passar dos anos e algumas das principais razões para isso são:

- **Estabilidade da equipe:** as equipes de desenvolvimento sempre ficam sujeitas a mudanças após a liberação do software, sendo que os membros podem ser realocados em novos projetos. Portanto, muitas vezes a equipe que desenvolveu o sistema não é a mesma equipe que realiza a manutenção, o que exige aumento nos esforços no processo de evolução.
- **Más práticas de desenvolvimento:** pode acontecer de os desenvolvedores não terem registrado em contrato a responsabilidade de manutenção, sendo assim, não há incentivo para construir o sistema de maneira que este seja preparado para mudanças futuras. Geralmente o contrato de manutenção é separado do contrato de desenvolvimento, o que abre a possibilidade de que a empresa que desenvolveu o software não seja a mesma que presta a manutenção, ou que sejam economizados esforços durante o desenvolvimento, aumentando os custos de manutenção.
- **Qualificações de pessoal:** geralmente as pessoas responsáveis pela manutenção, por muitas vezes serem membros de equipes diferentes

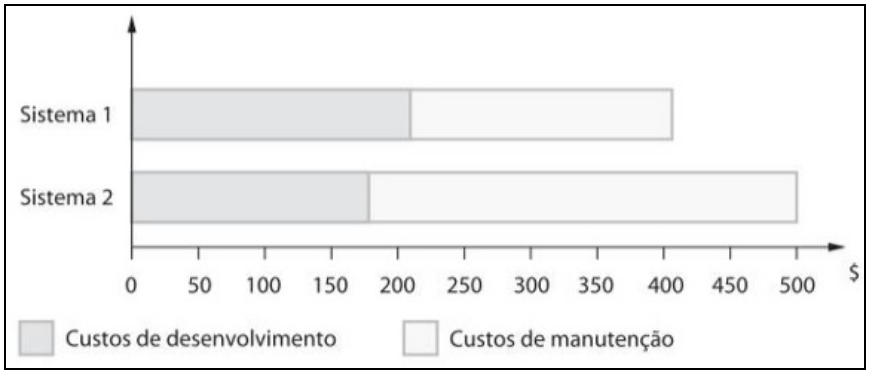
realocados, possuem pouco conhecimento e experiência sobre aquele determinado domínio de aplicação.

- **Idade do programa e estrutura:** com as alterações que vão sendo feitas no programa, sua estrutura tende a degradar, o que faz este software consequentemente envelhecer e mais difíceis de serem entendidos, dificultando alterações posteriores.

Como podemos analisar, as três primeiras razões apresentadas acontecem em muitas organizações pelo fato de considerarem o desenvolvimento e a manutenção tarefas separadas e sem relação uma com a outra. Já o quarto problema, refere-se a degradação natural do software com o passar do tempo.

Araújo, Souza e Vale (2011), afirmam que a dimensão dos esforços aplicados durante o desenvolvimento de um sistema, influenciam nos custos gerais durante a vida útil do software. Sendo assim, quanto maiores forem os esforços no desenvolvimento, menores serão os custos de evolução e manutenção do software (Figura 19).

Figura 19 - Custo de desenvolvimento e manutenção.



Fonte: Sommerville (2011, p.172).

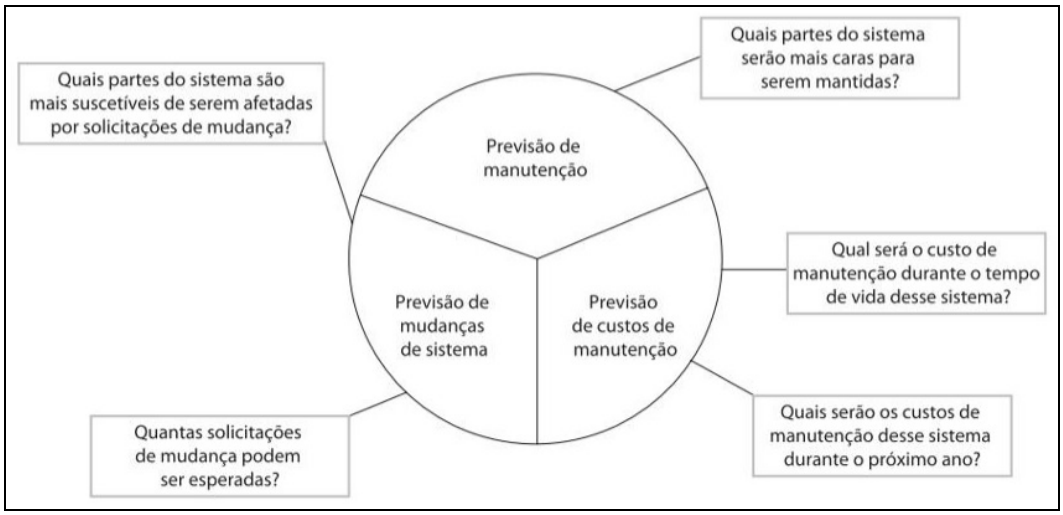
Uma maneira de reduzir os custos e as “surpresas” no período de evolução do software é trabalhar o desenvolvimento de software, desde a sua concepção, pensando em boas práticas que venham a aumentar a manutenibilidade do sistema em contexto.

Portella (2010), define manutenibilidade como o nível de facilidade com que um software pode ser entendido, corrigido, adaptado ou ampliado. E ainda indica algumas medidas da manutenibilidade de um software, ou simplesmente métricas de manutenção, sendo elas:

- Tempo de Reconhecimento do problema;
- Tempo de Análise do problema;
- Tempo de Especificação das mudanças;
- Tempo de Correção;
- Tempo de Testes;
- Tempo de Revisão da manutenção;
- Tempo de Recuperação total.

Tendo em vista estas métricas, surge a possibilidade de poder prever quais mudanças podem ser propostas e que partes do sistema serão, provavelmente, as mais difíceis de serem mantidas. Sendo assim, aumenta também a possibilidade de estimar os custos gerais com manutenção, evitando as “surpresas” indesejadas (Figura 20).

Figura 20 - Previsão de Manutenção.



Fonte: Sommerville (2011, p.173).

Segundo Sommerville (2011), prever o número de solicitações de mudança no sistema requer maior entendimento do relacionamento entre o sistema e seu ambiente externo, levando em consideração os seguintes fatores:

- **O número e a complexidade das interfaces de sistema:** quanto maior o número de interfaces e mais complexas elas forem, maior a probabilidade de serem exigidas as alterações de interface quando novos requisitos forem propostos.

- **O número de requisitos inerentemente voláteis ao sistema:** os requisitos que refletem as políticas e procedimentos organizacionais são provavelmente mais voláteis que os requisitos baseados em características estáveis de domínio.
- **Os processos de negócio em que sistema é usado:** como processos de negócios evoluem, eles geram solicitações de mudanças de sistema.

A engenharia de software busca utilizar melhores métodos de planejamento e desenvolvimento e um objetivo implícito desta busca parece ser que, com uma maior ênfase desenvolvimento, muitos dos problemas da evolução de software podem ser minimizados, especialmente no que se refere aos processos de manutenção.

2.5 VISÃO DA TECNOLOGIA ITIL

A ITIL (*Information Technology Infrastructure Library*) foi criada pelo governo britânico na década de 1980 com o objetivo de definir uma referência padrão para o gerenciamento de processos da área de TI de seus departamentos. No principio, foram estabelecidas melhores práticas que seriam utilizadas pelas organizações públicas para melhorar os resultados de custo e qualidade.

Segundo Pinheiro (2011), constitui-se de uma de uma descrição coerente e integrada de práticas de gerenciamento de serviços de TI, focando em pessoas, processos e recursos que são utilizados na entrega de serviços que atendam as necessidades dos clientes.

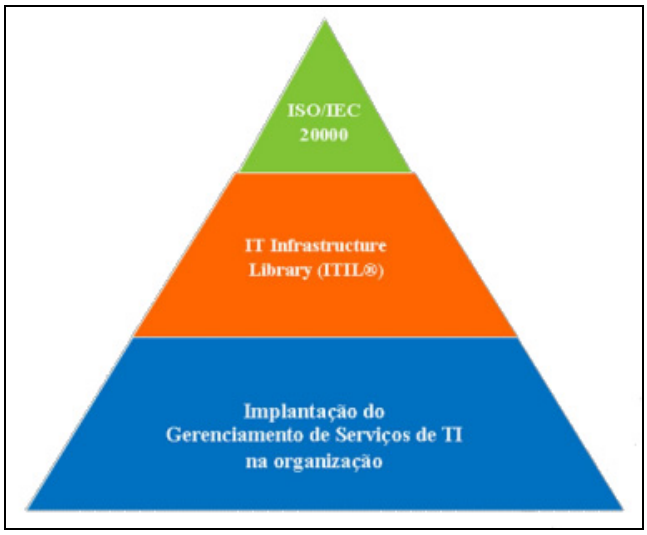
Desta forma, conforme a SISNEMA (2009), constitui-se como uma coleção das melhores práticas que tem como objetivo otimizar os serviços de TI e alinhar todos os componente com os requisitos do negócio, tendo como foco a qualidade e o atendimento ao cliente.

Tudo isso acontece, devido a grande dependência da tecnologia da informação para qualquer tipo de negócios hoje em dia, o que cria preocupação com o gerenciamento de serviços de TI nas empresas. Por exemplo, como imaginar, atualmente, a organização e gerenciamento de contas de um sistema bancário sem a utilização da tecnologia. Portanto, a área de TI deixa de ser pensada isoladamente somente por técnicos nas empresas, e passa a ser incorporada como um ativo de negócios.

“Mais de 10.000 empresas do mundo já adotaram as melhores práticas da ITIL, isto comprova sua maturidade e aceitação no mercado.” (SISNEMA, 2009, p.14).

De acordo com Pinheiro (2011), não existe certificação ITIL para empresas, apenas para profissionais. As empresas que quiserem obter um selo ou certificação para seus processos de TI poderão se certificar com base na ISO/IEC 20000 (Figura 21). A ITIL explica como devem ser os processos e a ISO/IEC 20000 tem os requisitos obrigatórios que especificam o que o provedor de serviços deve cumprir.

Figura 21 - Relação entre a ISO/IEC 20000 e a ITIL.



Fonte: Pinheiro (2011, p.11).

Conforme a SISNEMA (2009), as melhores práticas da ITIL têm como objetivos:

- Servir de inspiração para melhorar seus processos de TI;
- Sugerir onde é possível chegar, pois outras empresas já conseguiram resultados positivos;
- Sugerir para que servem os processos e práticas;
- Sugerir por que adotar os processos e práticas.

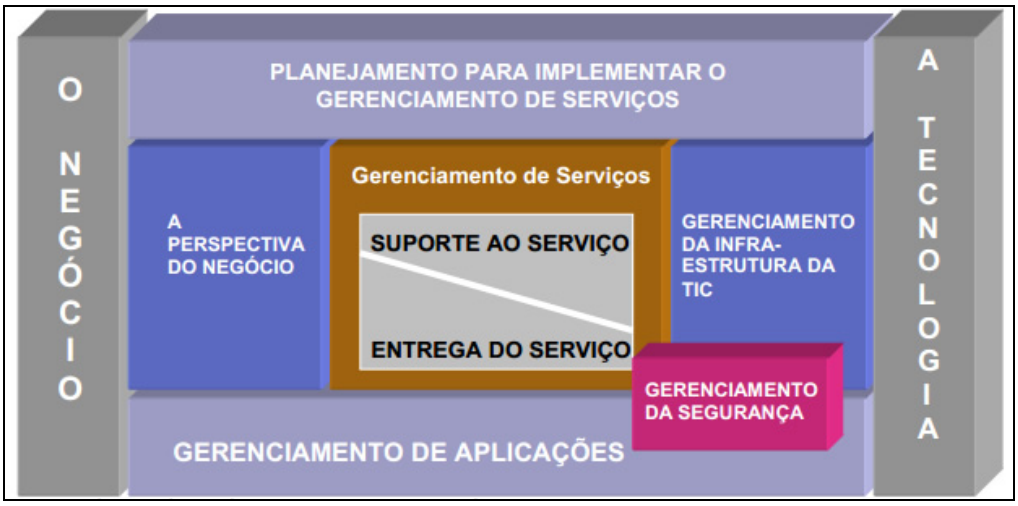
Seguindo estas melhores práticas a ITIL direciona o foco de TI aos negócios da empresa e presa para que a organização entregue os seus serviços aos clientes da melhor maneira possível a um custo justificável. Sendo assim, os gerentes de projetos possuem seu processo controlado e disponibilizado ao cliente abrangendo todos os fatores que vão determinar a qualidade de seu software.

De acordo com Damasceno, Araújo e Nunes (2009), a ITIL ajuda na criação de métricas para avaliar a capacidade de atendimento de demandas em uma empresa, medindo,

por exemplo, o tempo de entrega e o nível de satisfação entre todas as áreas de TI envolvidas no serviço em questão, criando um processo de melhorias contínuas.

"A ITIL oferece um *framework* comum para todas as atividades do departamento de TI, como a parte da provisão dos serviços, baseada na infra-estrutura de TI (Figura 22)." (SISNEMA, 2009, p.15).

Figura 22 - *Framework* da ITIL.



Fonte: SISNEMA (2009, p.16).

Conforme a Figura 22, podemos ver claramente como a ITIL associa os negócios da empresa com a tecnologia em prol do gerenciamento dos serviços, que fica rodeado de uma série de ações, procedimentos e metodologias de gerencia que vem a refletir na melhoria da entrega e no suporte do serviço.

Segundo a SISNEMA (2009), a biblioteca ITIL define os objetivos e atividades, as entradas e saídas de cada um dos processos encontrados em uma organização de TI. Porém, não proporciona uma descrição especifica de como estas atividades devem ser executadas, pelo fato de as etapas serem diferentes de uma organização para outra.

Em outras palavras a ITIL direciona e serve como referência, a partir de sugestões que foram provadas na prática, para o planejamento de processos mais comuns, papéis e atividades, fazendo a relação entre estes fatores e definindo as linhas de comunicação necessárias. Sendo assim, ITIL não é uma metodologia por não possuir uma implementação rígida.

Para ir de encontro com o que foi apresentado anteriormente, Pinheiro (2011) define como sendo características principais da ITIL os seguintes fatores:

- Modelo de referência para processos de TI não proprietário;
- Adequado para todas as áreas de atividade Independente de tecnologia e fornecedor;
- Um padrão de fato;
- Baseado nas melhores práticas;
- Um modelo de referência para a implementação de processos de TI;
- Padronização de terminologias;
- Interdependência de processos;
- Diretivas básicas para implementação;
- Diretivas básicas para funções e responsabilidades dentro de cada processo;
- Check List testado e aprovado;
- O que fazer e o que não fazer.

Conforme a SISNEMA (2009), a biblioteca ITIL organiza sua proposta de melhores práticas em uma série de cinco livros, que abordam os seguintes temas:

- **Suporte a Serviços:** descreve os processos ao suporte do dia-a-dia e atividades de manutenção associadas com a provisão de Serviços de TI.
- **Entrega de Serviços:** cobre os processos necessários para o planejamento e entrega de Serviços de Ti com qualidade e se preocupa ao longo do tempo com o aperfeiçoamento desta qualidade.
- **ICT – Gerenciamento da Infra-Estrutura:** cobre todos os aspectos do gerenciamento da infra-estrutura como a identificação dos requisitos do negócio, testes, instalação, entrega, e otimização das operações normais dos componentes que fazem parte dos Serviços de TI.
- **Planejamento para Implementação do Gerenciamento de Serviços:** examina questões e tarefas envolvidas no planejamento, implementação e aperfeiçoamento dos processos do gerenciamento de Serviços dentro de uma organização. Também foca em questões relacionadas à cultura e mudança organizacional.
- **Gerenciamento de Aplicações:** descreve como gerenciar as aplicações a partir das necessidades essenciais dos negócios, passando por todos os estágios do

ciclo de vida de uma aplicação, incluindo até a sua saída do ambiente de produção. Este processo da ênfase em assegurar que os projetos de TI e as estratégias estejam corretamente alinhados com o ciclo de vida da aplicação, garantindo que o negócio consiga obter o retorno do valor investido.

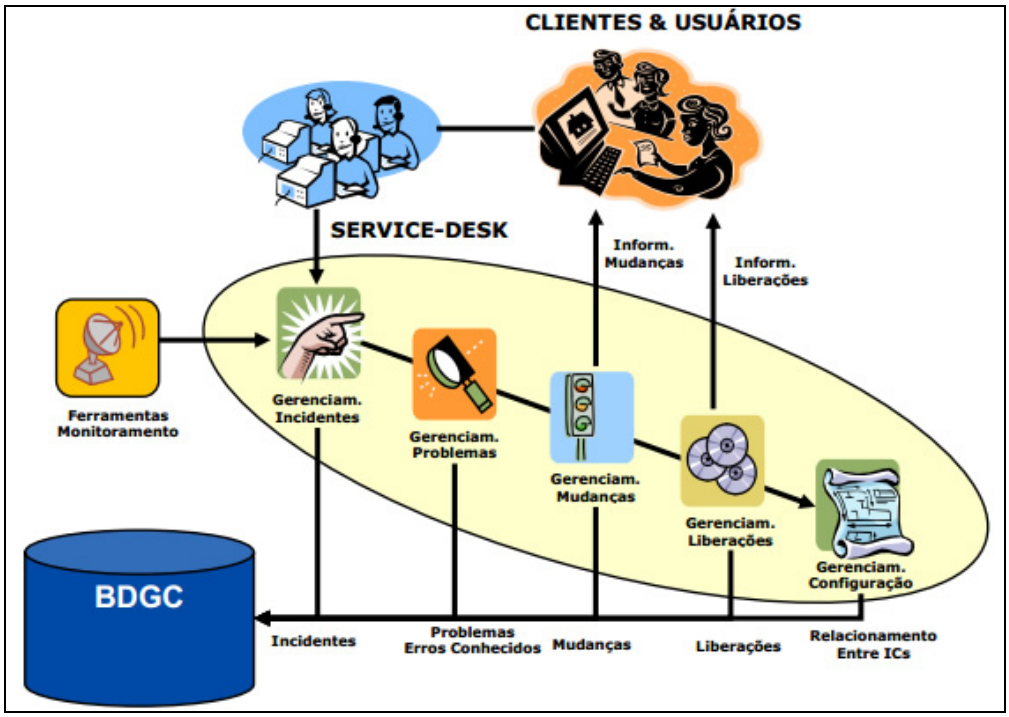
- **Perspectiva de Negócio:** fornece um conselho e guia para ajudar o pessoal de TI entender como eles podem contribuir para os objetivos do negócio e como suas funções e serviços podem ser alinhados e aproveitados para maximizar sua contribuição para a organização.
- **Gerenciamento da Segurança:** detalha o processo de planejamento e gerenciamento a um nível mais detalhado da segurança da informação e serviços de TI, incluindo todos os aspectos associados com a reação da segurança dos incidentes, a partir deste temos a concepção estratégias de segurança.

Porém, apesar da existência todos estes livros, cada um com um direcionamento especifico, a proposta de pesquisa deste trabalho esta mais alinha com o **suporte a serviços** e **entrega de serviços**.

De acordo com a SISNEMA (2009), o livro de suporte a serviços descreve como um cliente consegue acesso aos serviços para suportar seus negócios (Figura 23). Cobrindo os seguintes assuntos:

- Central de Serviços;
- Gerenciamento de Incidentes;
- Gerenciamento de problemas;
- Gerenciamento da Configuração;
- Gerenciamento das Mudanças;
- Gerenciamento de Liberação.

Figura 23 - Suporte a serviços segundo a ITIL.

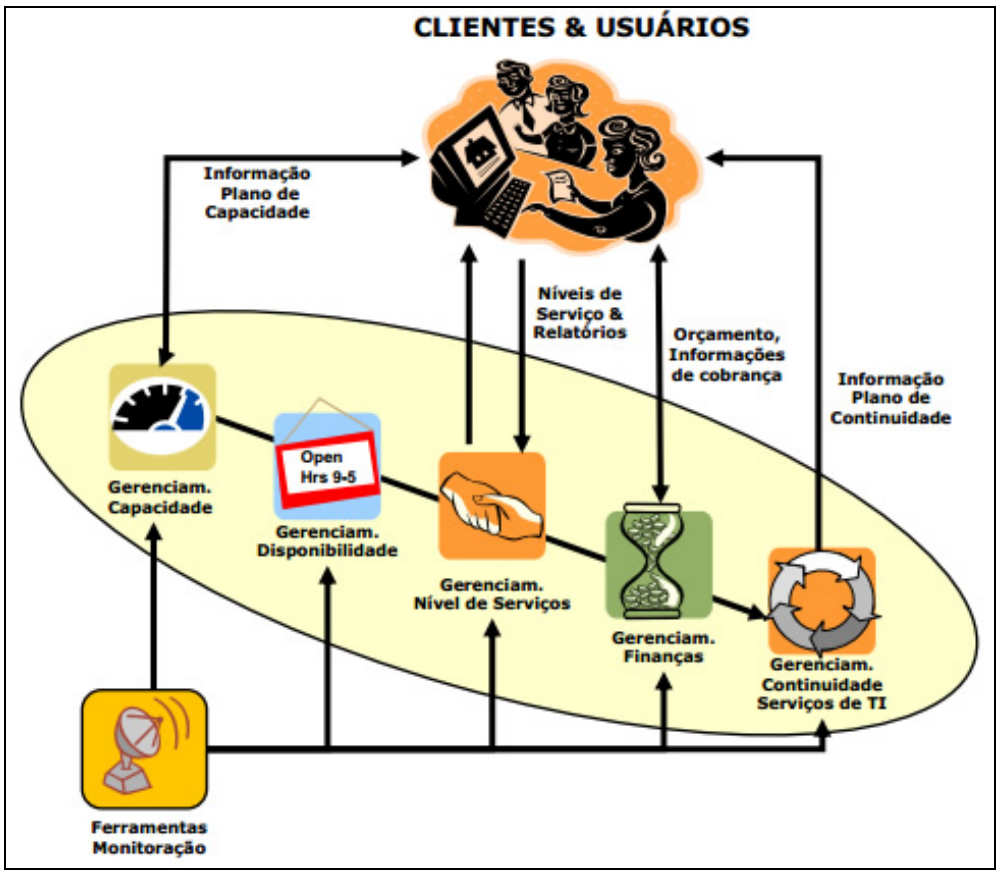


Fonte: SISNEMA (2009, p.20).

No livro a entrega de serviços, a SISNEMA (2009), descreve os serviços necessários que o cliente necessita, e o que é necessário para fornecer estes serviços (Figura 24). Este livro cobre os seguintes assuntos:

- Gerenciamento do nível de serviços;
- Gerenciamento financeiro para serviços de TI;
- Gerenciamento da capacidade;
- Gerenciamento da disponibilidade;
- Gerenciamento da continuidade dos serviços de TI;
- Gerenciamento da segurança.

Figura 24 - Entrega de serviços segundo a ITIL.



Fonte: SISNEMA (2009, p.21).

São diversos os componentes que, mediante sua análise, indicam o grau de maturidade em que uma organização se encontra quando da utilização das práticas de ITIL.

De acordo com GROFFE (2012), alguns dos principais componentes são:

- A maneira como a organização define um processo;
- A equipe, os papéis e as habilidades envolvidas;
- De que forma é possível medir ou comunicar o resultado do processo;
- Em que grau um processo está integrado aos outros processos e quão bem o mesmo está automatizado.

GROFFE (2012), ainda coloca dentre os benefícios que podem ser obtidos a parte da utilização das técnicas que compõem ITIL, os seguintes:

- Melhorias na satisfação dos clientes/áreas dependentes de um ou mais serviços;
- Maior eficiência operacional;

- Redução nos custos e nos esforços desprendidos pela área de TI cumprimento de uma ampla gama de atividades;
- Um suporte útil à organização para que esta efetue o ajuste de seus processos face à pressão exercida por normas regulatórias (como a lei Sarbanes-Oxley). Diante deste aspecto, é ainda possível a utilização em conjunto do ITIL com o COBIT;
- O alinhamento do setor de TI com a área de negócios. Neste ponto, os envolvidos da área de Tecnologia de Informação visualizarão seus serviços em termos de negócios e do que os mesmos representam para a empresa, não mais enxergando suas atividades apenas sob o espectro restrito da tecnologia.

Porém, apesar de tantos benefícios e ganhos que a organização pode ter utilizando o ITIL sempre vão existir problemas e dificuldades, uma vez que as empresas possuem aspectos específicos e particularidades quanto aos seus processos e serviços.

Desta forma, segundo a SISNEMA (2009), um projeto de implementação das práticas de ITIL pode ter alguns problemas, tais como:

- **Falta de patrocínio comprometimento e entendimento:** é crucial que todas as pessoas envolvidas com o projeto estejam conscientes das melhorias que a mudança poderá trazer. O comprometimento é fundamental para fazer as coisas realmente acontecer;
- **Cultura da empresa:** se a empresa não tiver cultura para gestão de serviços, se torna muito mais difícil a TI obter a colaboração dos demais departamentos;
- **Excesso de expectativa:** a adoção de melhores práticas não se faz em dias, sendo que é necessário muito planejamento, insistência, acompanhamento e adaptações ao longo do projeto de implantação;
- **Falhas na comunicação**;
- **Objetivos não alcançados:** objetivos como melhoria de qualidade, redução de custo, satisfação do usuário, alinhamento de TI com a estratégia de negócios.

Porém, conforme já foi dito, é importante frisar que o ITIL é nada mais é do que uma referência para auxiliar em uma mudança organizacional. Não pode ser definida como

metodologia. Sendo assim, cada organização deverá efetuar o planejamento de seus próprios processos tomando como embasamento os princípios do ITIL.

"Não é correto afirmar que um processo é '**compatível com ITIL**', nem mesmo implantar a ITIL. O objetivo é implantar o gerenciamento de serviços de TI, com base nas melhores práticas da ITIL." (SISNEMA, 2009, p.17).

3 ANÁLISE DO PROCESSO DE DESENVOLVIMENTO E SUPORTE A SERVIÇOS DE SOFTWARE NAS EMPRESAS DE PASSO FUNDO

Para garantir a qualidade e a confiabilidade de uma pesquisa utilizam-se sempre técnicas e metodologias científicas. Porém não existe uma metodologia de pesquisa padrão e aplicável a qualquer situação de análise de dados.

Desta forma, o que determina que metodologia de pesquisa se adéqua ao contexto é o tipo de estudo e o objetivo da análise.

No caso deste estudo do processo de software e suporte a serviços das empresas de desenvolvimento de Passo Fundo, o trabalho inicia na seleção de amostragem e segue na coleta e análise dos dados.

3.1 DELINEAMENTO DA PESQUISA

A metodologia desta pesquisa é de caráter qualitativo, quantitativo e exploratório, e tem por objetivo identificar, principalmente, como se apresenta o processo de software e processo de manutenção das empresas de desenvolvimento de Passo Fundo.

De acordo com Roesch (1996), a pesquisa qualitativa envolve uma instância teórica de maneira autoconsciente procura suspender suposições descuidadas sobre significados compartilhados. Procura o que é comum, mas permanece aberta para perceber a individualidade e os significados, em vez de destruí-los na busca por uma média estatística.

Já o método exploratório é definido por Diehl e Tatim (2004), como um método que objetiva proporcionar familiaridade com o problema, com o objetivo de torná-lo mais explicito ou construir hipóteses. Geralmente, envolve o levantamento bibliográfico, a realização de entrevistas com pessoas que possuem experiência prática com o tema.

Ainda segundo Diehl e Tatim (2004), o uso de quantificação no tratamento de informações caracteriza-se por meio de técnicas estatísticas, tais como, percentual, media, desvio de padrão, entre outras.

Mais especificamente, possui questões que direcionam a busca por alguns fatores como: o tipo de desenvolvimento mais abrangente no mercado (Desktop, Web, Mobile), as linguagens de programação mais utilizadas, organização da infra-estrutura das empresas em torno do atendimento ao cliente, metodologias utilizadas, se existe preocupação por parte dos gerentes de projetos quanto à melhoria de seu processo de desenvolvimento e com a qualidade de seus softwares, e como as empresas lidam com a manutenção e evolução de software.

Esta análise será uma ferramenta muito útil para projetos futuros voltados ao processo de software e ao processo de manutenção e evolução de software da região. No contexto deste projeto, as informações coletadas serão utilizadas para identificar pontos críticos na manutenção de software das empresas de desenvolvimento de Passo Fundo, e com base nestes estudos, associados à revisão bibliográfica, surgiram às considerações sobre o tema em discussão.

3.2 POPULAÇÃO E AMOSTRA

O público alvo selecionado para a pesquisa foram empresas associadas ao Polo de Exportação de Software do Planalto Médio (PoloSul), limitando-se apenas a organizações cidade de Passo Fundo.

Não foram utilizados critérios específicos para escolha dos atores envolvidos na pesquisa, mas apenas que estes obtivessem relação direta com o tema.

O número total de questionários respondidos dentre todos os solicitados a participar da pesquisa chegou a 16.

3.3 TÉCNICA DE COLETA DE DADOS

A estratégia de coleta de dados utilizada na pesquisa foi através da aplicação de questionário (Anexo I) com perguntas abertas e fechadas a partir dos objetivos de pesquisa do estudo.

Desta forma, para a efetuação da pesquisa foi utilizado um roteiro fechado e pré-definido a partir de contribuições teóricas já levantadas na revisão bibliográfica e, de dados em pesquisa de campo.

3.4 ANÁLISE E INTERPRETAÇÃO DOS DADOS

Após a aplicação do questionário, tem-se a análise e interpretação dos dados para filtrar e transformar em números as informações coletadas. Os documentos passaram por análise seguindo os objetivos do presente projeto, ou seja, serão organizados e revisados obtenção de conclusões e posicionamentos quanto aos resultados.

3.4.1 VARIÁVEIS

Para facilitar o estudo dos resultados, foram definidas algumas variáveis especificadas através do agrupamento dos objetivos das questões, sendo elas:

- **Caracterização das empresas estudadas:** serão dispostas as características das empresas estudadas, com a finalidade de identificar, mesmo que superficialmente, o perfil das empresas da cidade de Passo Fundo. Nesta variável, temos informações como: a quanto tempo estas empresas estão no mercado, quantos funcionários possuem, com que tipo de desenvolvimento trabalham, linguagens de programação utilizadas, quantidade de clientes que atendem e quanto satisfeitos acham que estão seus clientes.
- **Infra-Estrutura Básica:** busca por características da disposição lógica e física da na infra-estrutura interna da empresa. São apresentadas aqui informações como: se existe divisão de setores e como é realizada a organização desta infra-estrutura, levando em consideração o suporte a serviços.
- **Processo de Desenvolvimento de Software:** lista características da empresa quanto ao seu processo de desenvolvimento de software. São apresentadas informações como: definição do processo de software por parte dos gerentes, modelos de processos utilizados, grau de envolvimento da equipe com as metodologias implantadas, se é realizada documentação das etapas do processo e se utilizam ferramentas de apoio ao gerenciamento deste processo.
- **Processo de Evolução e Manutenção de Software:** descreve as características das empresas quanto o atendimento aos seus clientes, após a entrega de seus softwares. Levando em consideração fatores como: se as empresas conseguem associar os processos de manutenção com os processos normais de desenvolvimento, ou seja, associar o suporte a serviços com as novas demandas, quanto as empresas se preocupam com a evolução do software, se utilizam técnicas de gerencia da configuração (versionamento), se o suporte a serviços encontra-se separado do desenvolvimento, se existe a utilização de software *help desk*, disposição das etapas para atendimento ao cliente, se as demandas de manutenção são tratadas conforme o tipo, ou seja, se estas

demandas são classificadas antes de serem tratadas e o que acreditam que possa ser melhorado quanto ao suporte.

- **Melhoria Contínua:** ressalta que tipos de ações as empresas tomam quanto a melhoria contínua e a busca pela qualidade de seus softwares. Argumentando sobre aspectos como: o que consideram mais importante para ter um produto de qualidade, quantas empresas dizem realmente trabalhar a melhoria de seu processo de desenvolvimento, o que consideram sendo a principal medida a ser tomada ou fator determinante para a melhoria e qual a visão quanto a modelos de maturidade de processos.

3.5 CONSIDERAÇÕES E DISCUSSÕES DE RESULTADOS

O estudo aplicado nas empresas de desenvolvimento de Passo Fundo possibilitou a criação de uma visão geral sobre o que está sendo utilizado no mercado local e qual o direcionamento destas empresas quanto ao futuro.

O número total de organizações que participaram e colaboraram para pesquisa foi de 16 empresas. Este número é satisfatório, levando em consideração a quantidade de empresas da cidade que possuem condições de somar informações relevantes para a pesquisa, devido ao seu tamanho e quantidade de funcionários, e levando em conta também o fato de que as empresas não costumam abrir informações sobre seu processo de desenvolvimento, estratégias de negócio e organização interna.

A média de idade das empresas entrevistadas chegou a 12,5 anos, sendo que tem-se a empresa a mais tempo no mercado com 45 anos e a que está a menos tempo completou 2 anos. Desta forma, pode-se perceber que foram estudadas empresas que, teoricamente, deviam possuir um nível de maturidade e estabilidade considerável no mercado, pois em média todas são organizações que passaram dos primeiros 5 anos de afirmação no mercado.

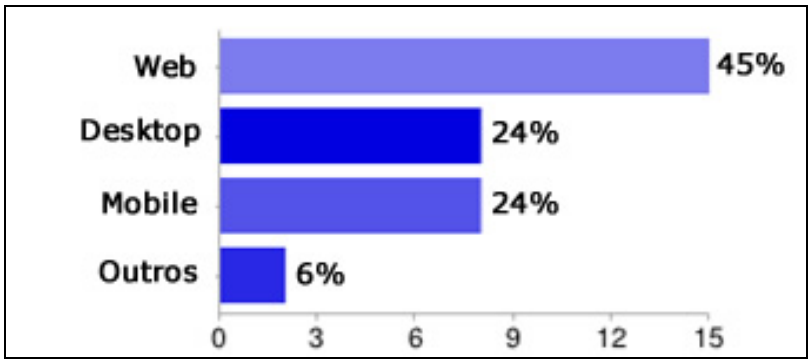
De maneira geral, não foram identificadas empresas de desenvolvimento que possam ser consideradas grandes em Passo Fundo. Existem casos de empresas grandes que se instalaram aqui, mas no contexto do mercado da cidade, todas as entrevistadas são empresas de pequeno a médio porte, com equipes de desenvolvimento pequenas.

A maior quantidade de funcionários encontrada foi de 120 funcionários, mas como dito anteriormente, não estão todos alocados em Passo Fundo. A menor quantidade foi de 2 funcionários. A média de funcionários é de 20 funcionários por organização.

Em relação à quantidade de clientes, tem-se uma média de 120 clientes por empresa, porém, com algumas específicas que fogem do padrão, no caso das empresas que apenas instalaram filiais em Passo Fundo. Sendo assim, foi encontrada a quantidade de 3000 clientes como sendo a maior e 7 clientes a menor quantidade registrada.

Seguindo a caracterização das empresas, foi identificado que a maior parte delas está voltada para o desenvolvimento web, onde 45% encontram-se voltadas para este nicho. Além disso, vale ainda ressaltar o significativo crescimento do desenvolvimento *mobile*, que alcançou 24% do total, igualando-se ao desenvolvimento *desktop*. Com o restante, ficam 6% das empresas que trabalham com outros tipos de desenvolvimento (Figura 25).

Figura 25 - Tipos de Desenvolvimento.

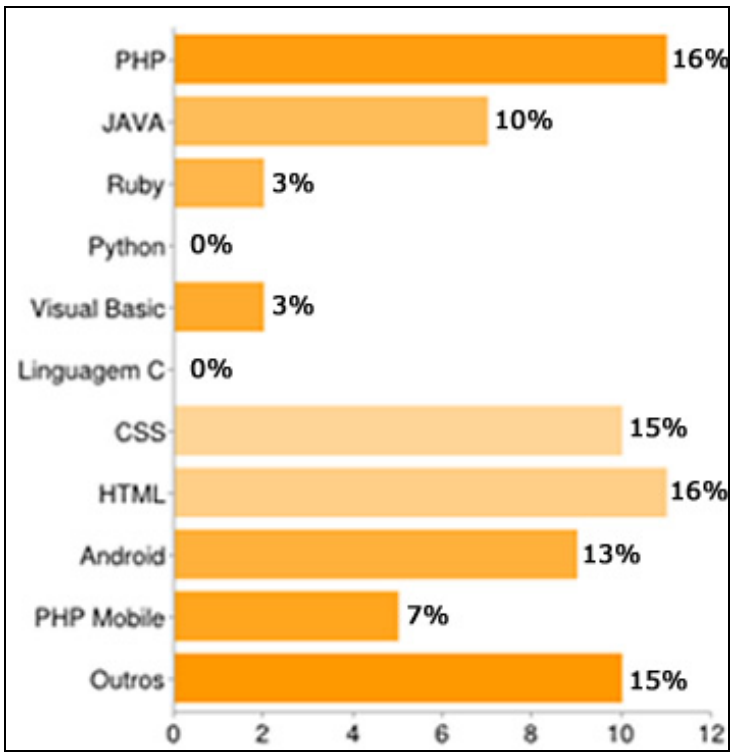


Fonte: Do Autor.

Quanto a linguagens de programação, as que mais aparecem no mercado passo fundense são linguagens de programação para web e mobile, dando destaque para o PHP e o HTML, que são utilizadas pela maioria.

Conforme a figura 26, temos as linguagens dispostas nas empresas da seguinte maneira: PHP (16%), HTML(16%), CSS(15%), outras linguagens (15%), Android (13%), JAVA (10%), PHP Mobile (7%), Visual Basic (3%), Ruby (3%), Python e Linguagem C (0%).

Figura 26 - Linguagens de Programação utilizadas.



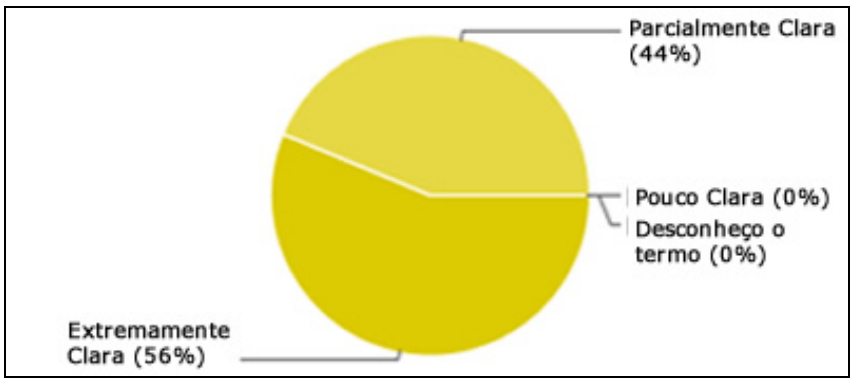
Fonte: Do Autor.

Quando questionadas sobre o nível de satisfação de seus clientes, aproximadamente 80% das empresas dizem que seus clientes estão totalmente satisfeitos. Foi identificada uma segurança grande por parte das entrevistadas. Porém, o excesso de confiança pode causar comodidade, afetando assim a capacidade de crescimento da empresa, sendo assim, é importante que os gestores de projetos desse mercado, além das equipes de desenvolvimento, comecem a pensar e desenvolver softwares com a visão do usuário, colocando-se no lugar de quem opera seu software, para depois poder afirmar com tanta certeza a satisfação absoluta de seus clientes.

Tudo isso pode ser averiguado quando avaliamos a variável de processo de desenvolvimento de software. Afinal, a organização e maturidade do processo de desenvolvimento refletem na qualidade do produto entregue, o que vai ser um dos fatores a determinar a satisfação dos clientes.

Os gestores foram questionados quanto sua definição ou entendimento sobre o termo processo de software e 56% deles dizem ter uma definição clara e 44% afirmam ter uma definição parcialmente clara (Figura 27).

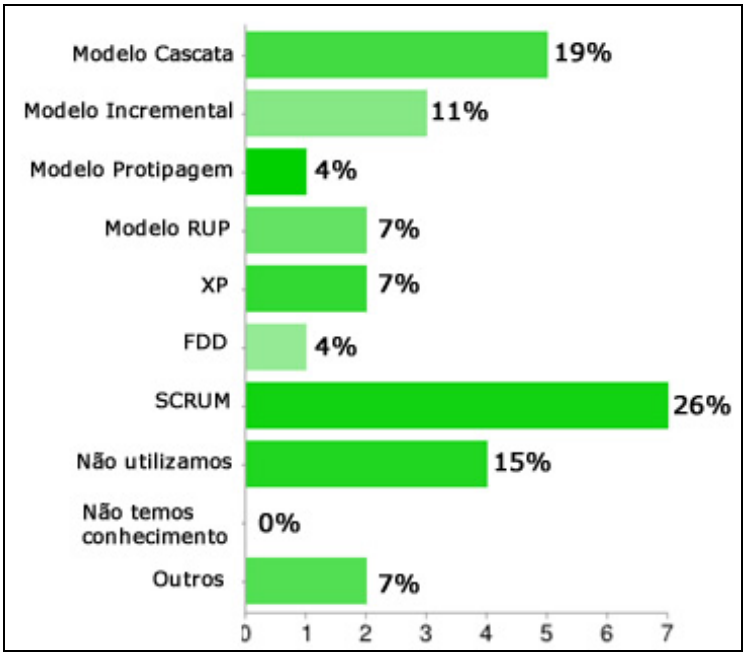
Figura 27 - Definição do termo processo de software.



Fonte: Do Autor.

São números satisfatórios que comprovam que os gestores possuem pelo menos o mínimo de conhecimento do que se trata o processo de software, porém, a maneira como eles implantam e fazem acontecer este processo na empresa, abre outro questionamento. Sendo assim, desde a menor empresa ou com menor quantidade de funcionários, até a maior das entrevistadas, os resultados garantem que vai existir, no mínimo, um processo básico ou uma sequência de tarefas organizadas.

Figura 28 - Metodologias de processos utilizadas.



Fonte: Do Autor.

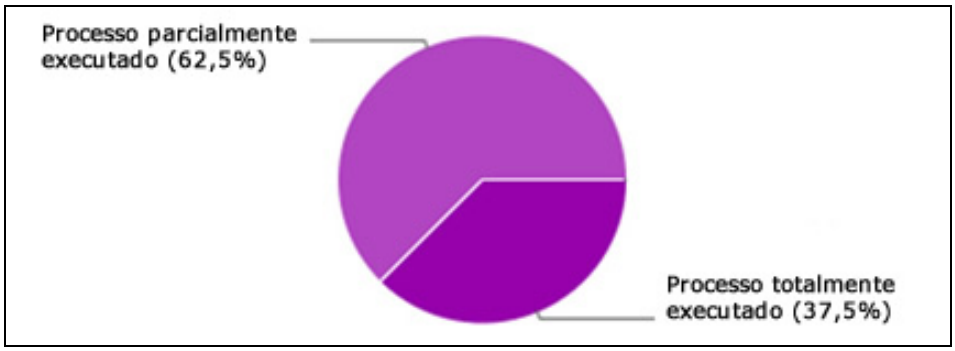
As metodologias ágeis de processos lideram o mercado de Passo Fundo com o Scrum, por outro lado, vemos uma grande quantidade de empresas utilizando metodologias um tanto

ultrapassadas, caso do modelo tradicional em Cascata (Figura 28). Outro número preocupante neste fator é os 15% de empresas que dizem não utilizar nenhuma metodologia de processos.

Conforme o gráfico da figura 28: SCRUM (26%), Modelo Cascata (19%), Não utilizam (15%), Modelo Incremental (11%), Modelo RUP (7%), XP (Extreme Programming) (7%), Outros Modelos (7%), Prototipagem (4%), FDD (4%), sem conhecimento do assunto (0%).

Com a finalidade de verificar a integridade dos dados levantados na ultima característica citada, as entrevistadas foram questionadas sobre como consideravam o grau de participação, envolvimento e disciplina dos integrantes da equipe em relação à metodologia de processos implantada. Este quesito mostra que as empresas se contradizem nestas questões, pois 37,5% delas dizem que o processo é totalmente executado por todos os integrantes da equipe e os outros 62,5% dizem que a metodologia é parcialmente executada (Figura 29).

Figura 29 - Envolvimento da equipe com as metodologias utilizadas.



Fonte: Do Autor.

Desta forma, não fica claro onde se enquadram os 15% de empresas que não utilizam nenhuma metodologia de processo de desenvolvimento, citadas na questão anterior.

Com isso, pode-se notar que apesar de os gestores terem uma posição definida quanto ao conceito de processo de software, não são todos que conseguem ter controle do processo e fazer acontecer na prática.

As razões para isso pode ser o fato de as empresas pensarem, primordialmente, em produzir mais, atender cada vez mais demandas do que pensar na organização de seu processo. Como se diz popularmente, “as empresas querem cortar cada vez mais lenha, mas não param para afiar o machado”, ou seja, não param para analisar seu processo e decidir o que está bom, para definir padrões de desenvolvimento, e onde se pode melhorar.

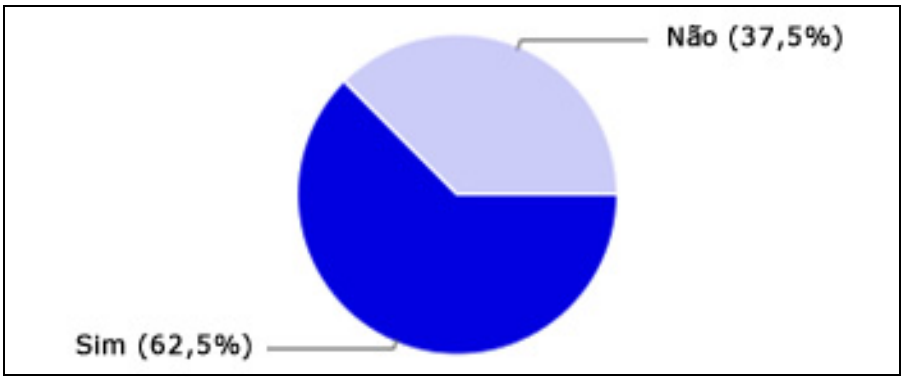
O ganho de produtividade é muito grande quando a equipe busca esse tempo para reflexão e análise sobre o que está sendo feito, sobre o produto que está sendo entregue. Com isso, pode-se definir padrões de desenvolvimento da empresa (nomenclaturas, estruturação de diretórios dos softwares, gerencia adequada do tempo, entre outros), além de que todos passam a ter conhecimento de tudo o que esta sendo produzido, sabem de que maneira se comporta seu produto.

Além do mais, não existe uma metodologia que seja melhor que outra, depende de empresa para empresa, de projeto para projeto. Portanto, é aconselhável a empresas de pequeno a médio porte, com equipes pequenas, caso das organizações de Passo Fundo, que procurem estudar as metodologias do mercado, informar-se sobre estudos de caso e exemplos de implantações, procurando talvez extrair o que uma ou outra apresenta de mais útil para seu contexto, e através desta análise, combinada com o conhecimento da capacidade de sua equipe e com a realidade de sua organização definir sua própria metodologia, potencializando o seu diferencial.

Isso se reflete em melhores condições de custo e beneficio, não somente durante o ciclo normal de desenvolvimento do software, mas também após a sua entrega, onde é comprovado que estão os maiores gastos das organizações. Possivelmente esses softwares terão necessidades de alterações menos frequentes e quando solicitadas essas futuras alterações serão bem mais fáceis de serem manipuladas.

Interessante saber que a maioria das empresas realiza documentação as etapas de seu processo, totalizando 62,5% que documentam e 37,5% que não descrevem nada (Figura 30).

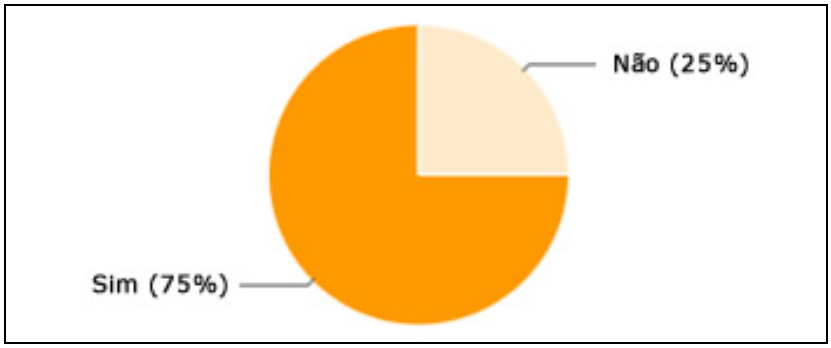
Figura 30 - Documentação das etapas do processo de desenvolvimento.



Fonte: Do Autor.

Além disso, 75% das entrevistadas utilizam software ou alguma ferramenta de apoio à documentação e gerenciamento do seu processo. Outras 25% não se preocupam com isso (Figura 31).

Figura 31 - Utilizam software ou ferramenta de apoio a gerencia do processo.



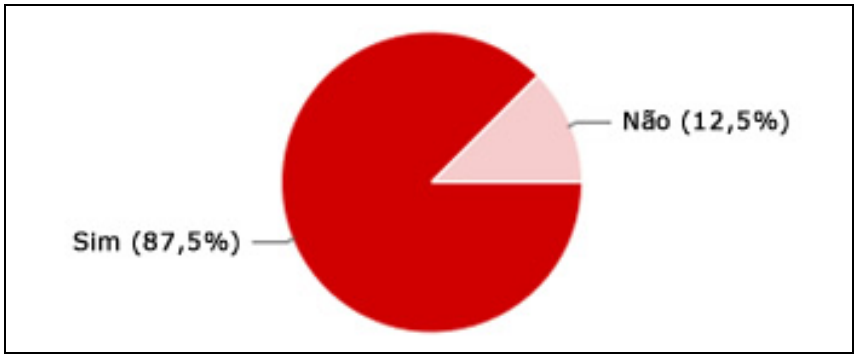
Fonte: Do Autor.

As metodologias ágeis são as mais utilizadas e mais fáceis de serem implantadas pelo fato de não exigirem muita documentação e não se apresentarem tão burocráticas, quanto às tradicionais, sendo as mais adequadas a empresas com o perfil das entrevistadas em Passo Fundo.

Portanto, pensando na evolução do projeto é interessante possuir algum tipo de documentação, por mais escassa que ela venha a ser, sempre é útil. Podem ser documentos encontrados no cliente, casos de uso, diagrama de classes, registro de tarefas em algum software ou ferramenta, por exemplo.

Apesar de encontramos empresas de diversas proporções, a maioria delas, aproximadamente 87,5%, possuem divisão lógica ou física de setores. Outros 12,5% das empresas, que são empresas menores, não realizam esta divisão (Figura 32).

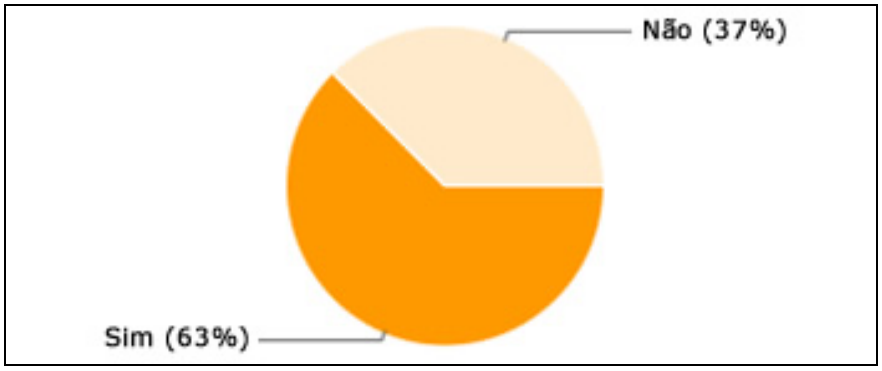
Figura 32 - Divisão por setores.



Fonte: Do Autor.

10 empresas é o número das que dizem possuir setor de suporte e manutenção de software, totalizando 63% das organizações e as outras 37% não possuem setor de suporte a serviços (Figura 33). Dentre estas 10 empresas citadas, 8 delas possuem o suporte a serviços separados do desenvolvimento.

Figura 33 - Apresentam Suporte como setor independente.



Fonte: Do Autor.

Sendo assim, as que possuem o suporte separado do desenvolvimento, estão a um passo de implementar uma das principais práticas estabelecidas pela ITIL, o conceito de *Service Desk* ou Central de Serviços.

"A Central de Serviços é uma função dentro da TI que tem o objetivo ser o ponto único de contato entre os usuários/clientes e o departamento de TI." (SISNEMA, 2009, p.29).

É uma proposta válida para as empresas estudadas, pois, segundo a SISNEMA (2009), a central de serviços sugere justamente separar dentro das operações de TI quem faz parte do suporte aos usuários de quem vai realizar atividades de resolução de problemas e desenvolvimento.

Desta maneira, surgem vantagens para os usuários, uma vez que o suporte será realizado com maior agilidade e qualidade, e também para a equipe de desenvolvimento que não precisará mais ser interrompida pelas chamadas diretas dos usuários a todo o momento.

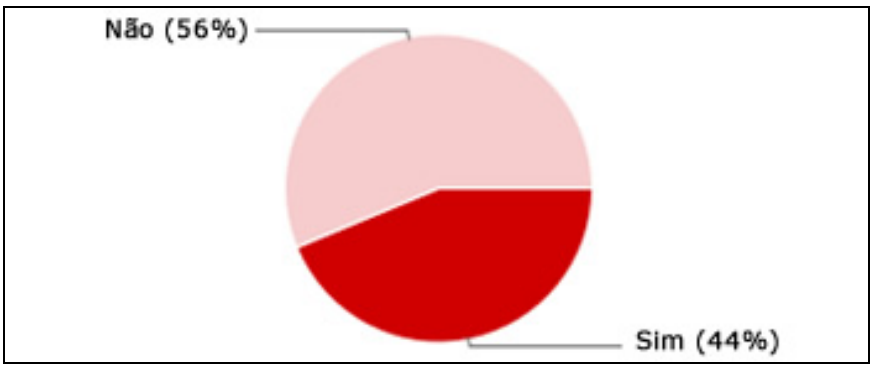
É uma forma interessante de organizar o suporte a serviços, porém, no caso das empresas entrevistadas, onde temos a maioria voltada para desenvolvimento web, terá de existir um meio termo nesta idéia. Afinal, o desenvolvimento web necessita de agilidade, sendo que a comunicação direta é uma forma muito comum de resolver rapidamente problemas, sendo uma das práticas defendidas pelo processo do XP, portanto, vão existir casos em que a comunicação direta de membros da equipe com o cliente será necessária.

Sendo assim, no contexto de organização das empresas de Passo Fundo, a proposta de criação de central de serviços, seria no sentido de reduzir a quantidade de intervenções diretas, mas não extinguir completamente a prática deste diálogo mais próximo, entre clientes e desenvolvedores, que é muito útil quando pensa-se em agilidade de negócios.

Nas organizações que possuem equipes menores é complicado alocar um setor com pessoas para trabalhar apenas com suporte. Porém, existem outras soluções para auxiliar na gerencia de processos de manutenção, por exemplo, a utilização de software de *Help Desk*.

Quanto à utilização de software de *Help Desk*, 56% das entrevistadas dizem não utilizar nenhuma ferramenta para tal finalidade, enquanto 44% utilizam algum tipo de software com este objetivo (Figura 34). Alguns utilizam softwares adquiridos através de terceiros e outros desenvolveram seus próprios sistemas de *Help Desk*.

Figura 34 - Utilizam software de *Help Desk*.



Fonte: Do Autor.

Conforme a SISNEMA (2009), a central de serviços é uma evolução do conceito de *Help Desk*, pois, um *Help Desk* tradicionalmente atende e serve como meio de comunicação com o cliente para resolução de problemas básicos. Por outro lado, a central de serviços na ITIL, assume todas as solicitações dos usuários relacionadas a qualquer serviço prestado.

Grande parte das empresas estudadas, ainda gerencia as mudanças de modo informal, percebe-se isso quando foram questionadas sobre como organizavam o atendimento ao cliente, desde a entrada de uma solicitação, até o fechamento da tarefa. As empresas realizam atendimentos sem definição de etapas, em alguns casos, tratando quase todas as solicitações de mudança da mesma maneira, ou em um mesmo nível, através de atendimentos pelo telefone, *messenger*, email, *skype* e outras mídias sociais, sem documentar quase nada e, geralmente, quem trata essas entradas são os próprios membros da equipe de desenvolvimento.

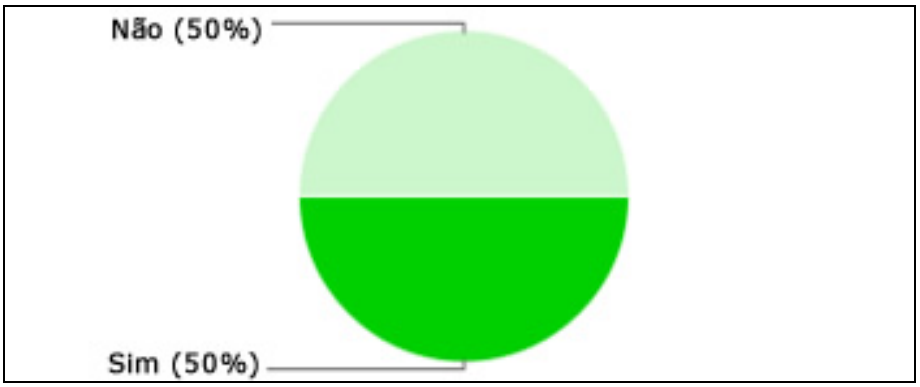
Sendo assim, pode estar neste contexto o principal motivo de grande parte das equipes não conseguirem associar as demandas de manutenção, com as novas demandas de software, como citado mais adiante, o que vem a causar problemas com cronogramas e, consequentemente, perda de qualidade do produto desenvolvido.

Percebe-se também que as empresas que se utilizam de softwares de apoio aos processos de manutenção e *Help Desk*, apresentam maior facilidade em ordenar as tarefas de manutenção em etapas bem definidas.

A utilização de um software com esta finalidade é uma boa proposta para organizar o suporte a serviços, pois estes softwares de *Help Desk*, possibilitam organizar as entradas de solicitação de mudanças, direcionar as atividades a uma pessoa ou grupo de pessoas mais habilitada a resolução do problema, definir níveis de atendimento, classificar as demandas, auxiliam na definição de planos ou contratos de suporte que possibilitem controlar a quantidade de demandas por cliente, podendo assim cobrar a manutenção com maior coerência, além de documentar todas as ações, interações, conversas, tempo de realização dos atendimentos.

Tendo essas variáveis documentadas através de um software de apoio, como citado anteriormente, os gerentes de projeto tem em mãos métricas importantes para estudar e definir a capacidade da sua equipe de desenvolvimento, ou seja, saber a quantidade de atividades, sempre considerando também o grau de complexidade das tarefas, que podem ser realizadas em determinado período de tempo, aumentando assim o controle sobre todo seu processo de desenvolvimento.

Figura 35 - Classificam por tipos de manutenção.



Fonte: Do Autor.

Organizar o atendimento ao cliente de acordo com o tipo de manutenção também é uma boa maneira de prover soluções no suporte a serviços. No caso das empresas

entrevistadas, 50% delas garante classificar o atendimento por tipo de manutenção (Figura 35).

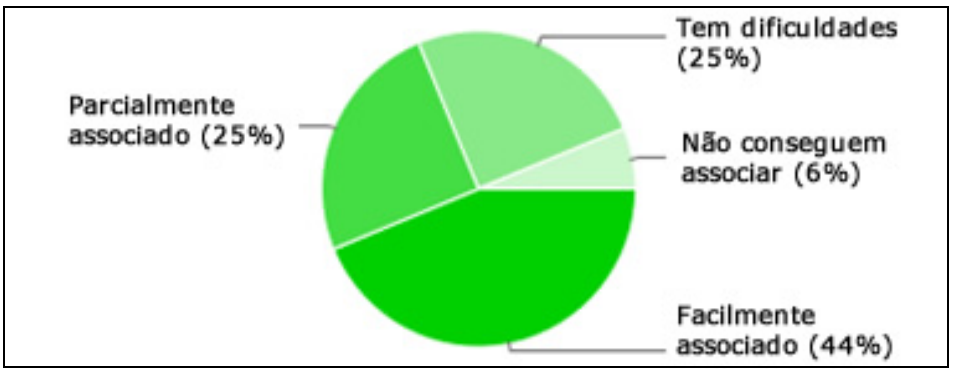
A disposição dos atendimentos classificando as tarefas por tipo de manutenção (Corretiva, Adaptativa, Perfectiva e Preventiva) permite que sejam criados níveis de atendimento. Para empresas de pequeno a médio porte, poderiam ser criados, por exemplo, três níveis de atendimento:

- **Nível I:** atendimento de demandas mais leves, como alterações de conteúdo, pequenas alterações visuais, entre outros. Enfim, tarefas que não exijam um grau de conhecimento técnico tão elevado, podendo ser realizado por um estagiário, por exemplo. Seria interessante que todas as demandas de suporte passem por esta pessoa, desta forma, esta sendo criada uma rotina, que futuramente, com o crescimento da empresa e aumento da quantidade de funcionários, pode se transformar na tão discutida central de serviços.
- **Nível II:** caso o Nível I não esteja apto atender as alterações solicitadas, os integrantes devem passar as solicitações ao Nível II, onde seria dado o apoio técnico do setor de produção. Neste nível a demanda é passada para própria equipe de desenvolvimento.
- **Nível III:** se o nível II julgar que a demanda é muito complexa e seu impacto é grande sobre o software em funcionamento. Esta deve ser passada para o nível III, que seria o setor comercial e de análise, que vai avaliar a solicitação e, se necessário, orçar esta demanda e estudar as possibilidades junto ao cliente.

Caso esta ainda seja uma solução complicada de ser implantada, o melhor a fazer é gerenciar o tempo da própria equipe de desenvolvimento, ou seja, definir períodos do dia para trabalhar apenas no atendimento e suporte ao cliente, dividindo as atividades diárias. Um momento atende as novas demandas, em outro presta suporte.

Estas soluções propostas podem servir para auxiliar em um problema encontrado nos resultados do questionário que é a dificuldade que as empresas possuem para associar as atividades do processo de manutenção, com as atividades das novas demandas da empresa (Figura 36).

Figura 36 - Associação das demandas de manutenção com as novas demandas da empresa.



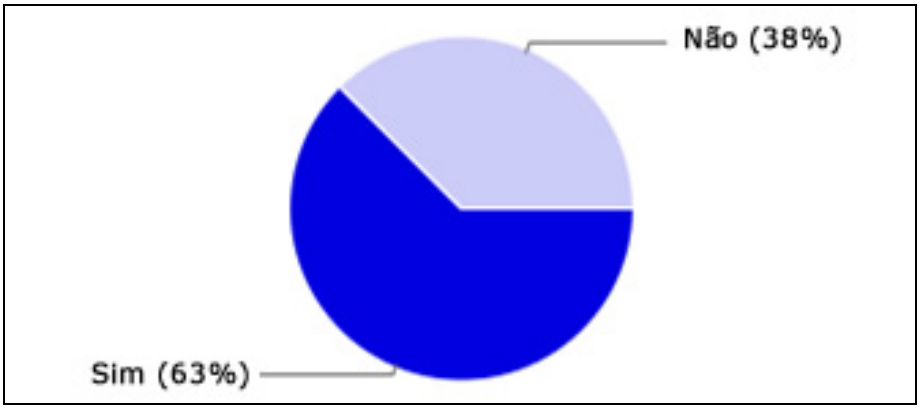
Fonte: Do Autor.

Conforme o gráfico da figura 36, apenas 44% das empresas conseguem associar facilmente as atividades, 25% associam parcialmente, outros 25% tem dificuldades claras para associar e 6% realmente não conseguem associar as atividades e com isso tem problemas com cronogramas e tempo de alguns projetos.

Como já foi citado anteriormente, documentar etapas de processo de software é uma prática fundamental para qualquer tipo de desenvolvimento, não diferente para os processos de manutenção de software. Documentar solicitações do cliente e alterações realizadas é uma ação é útil para evitar problemas e para auxiliar em futuras alterações. O gerenciamento de versões, neste caso, é uma boa solução.

Quanto a este fator, foi identificado que 63% das entrevistadas utilizam software para auxiliar na gerencia das mudanças ou de técnicas de versionamento de software. Enquanto outros 38% não utilizam nada neste sentido (Figura 37).

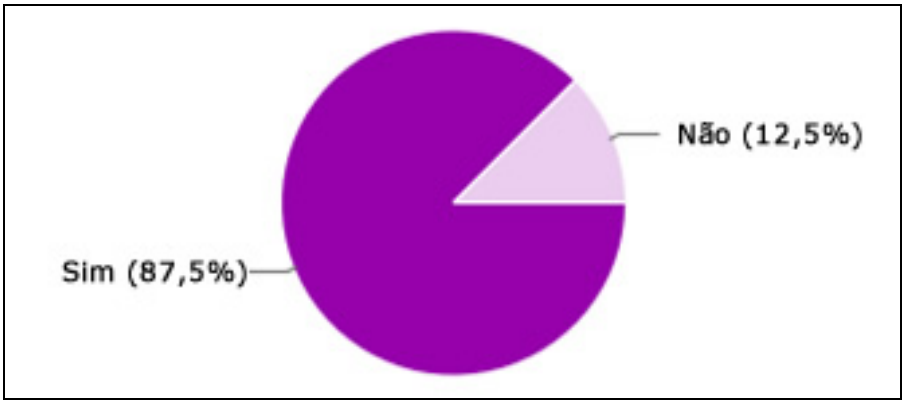
Figura 37 - Utilizam gerenciador de versões.



Fonte: Do Autor.

Apesar das inconformidades encontradas até o momento, quanto a definição dos processos de manutenção, por parte das entrevistadas, 87,5% delas dizem se preocupar com a evolução dos softwares que desenvolvem e apenas 12,5% não pensam se preocupam com a evolução (Figura 38).

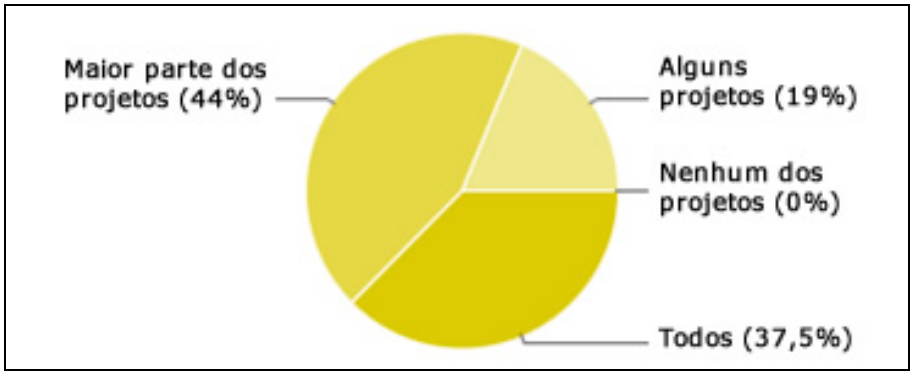
Figura 38 - Se preocupam com a evolução do software.



Fonte: Do Autor.

Além disso, para reforçar esta idéia, 37,5% das empresas dizem desenvolver todos os projetos procurando prever possíveis solicitações de mudança, 44% dizem realizar isso na maioria dos projetos e 19% em alguns projetos apenas (Figura 39).

Figura 39 - Programam pensando em possíveis alterações futuras.



Fonte: Do Autor.

Quanto a que as organizações estudadas acreditam que podem melhorar em seu suporte a serviços, foram identificados os seguintes fatores, em ordem de maior ocorrência:

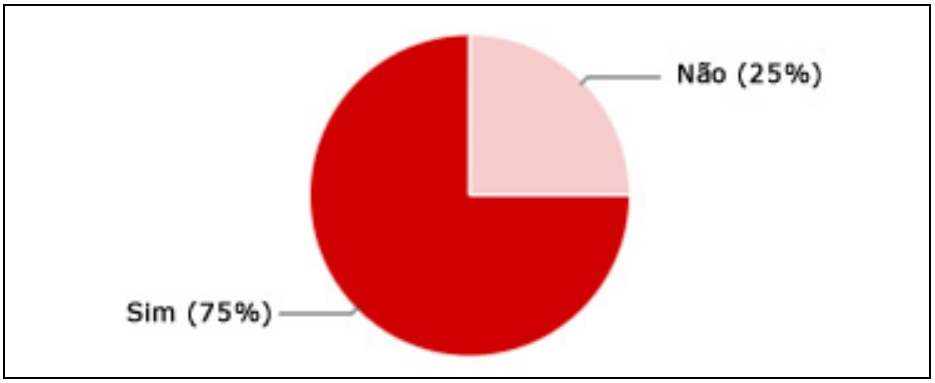
- Separar o suporte a serviços do desenvolvimento;

- Buscar *feedback* mais constante sobre a satisfação do cliente e os resultados obtidos por ele utilizando o produto;
- Implantação de software de *Help Desk*;
- Utilização das práticas da ITIL;
- Melhorar o processo de desenvolvimento para reduzir demandas de suporte;
- Melhorar a prestação de treinamento;
- Divisão das demandas por tipo de manutenção;
- Documentar atendimentos.

Através dos últimos parâmetros apresentados, nota-se que os conceitos, a teoria e o mínimo de direcionamento correto as empresas e seus gerentes de projetos possuem, em alguns momentos até indo de encontro as soluções propostas, porém, a maior dificuldade encontrada esta em executar as suas ambições e transformar em ações.

Uma das questões primordiais para se alcançar qualidade de software é a melhoria contínua dos processos envolvidos no desenvolvimento. Em relação a este fator, 75% das entrevistadas afirmam trabalhar na melhoria de seu processo de desenvolvimento (Figura 40).

Figura 40 - Trabalham a melhoria contínua de seu processo.



Fonte: Do Autor.

Segundo Sommerville (2011), a maturidade do desenvolvimento de software em uma empresa alcança seu maior nível quando seus processos são continuamente melhorados, baseados, entre outros aspectos, num retorno (*feedback*) quantitativo (mensurável) dos processos.

Porém, o problema é que, geralmente, depois do software estar pronto e ser implantado no ambiente do usuário não existe um acompanhamento para avaliar se o software

está realmente atendendo as expectativas. Nenhum *feedback* existe para o desenvolvedor, exceto quando ocorre algum problema grave e é necessária alguma intervenção, ou alteração devido a fatores externos, como por exemplo, uma nova legislação. Sendo assim, as estatísticas de uso do software, no geral, são escassas.

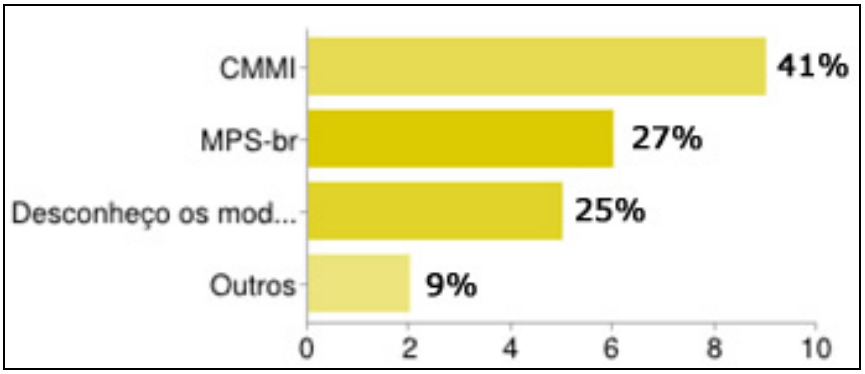
Ainda pensando em qualidade de software, foi identificado que 69% das empresas de desenvolvimento de Passo Fundo acreditam que o fator determinante da qualidade de seus produtos, é o conhecimento das necessidades do cliente, uma definição clara do escopo do projeto, que cubra todos os requisitos antes do início do desenvolvimento.

Foram citados também outros aspectos como definição de padrões de codificação, grau de capacidade técnica da equipe e qualificação do produto através de testes.

A visão das empresas quanto às estes parâmetros está correta, principalmente no que diz respeito a definição do escopo do projeto, mas ainda é uma visão muito funcional de suas atividades. As empresas necessitam pensar mais em alinhar a tecnologia da informação com os negócios da empresa, desta forma, poderiam ser considerados também fatores pela busca da qualidade uma boa estruturação do suporte a serviços, documentação adequada das atividades do projeto, planejamento e reflexão sobre o processo atual, gerência adequada do tempo em relação à quantidade de demandas e aplicação de conceitos de usabilidade, por exemplo.

Para auxiliar as organizações na busca pela qualidade e melhoria de seu processo, existem os modelos de maturidade de processos, sendo os mais conhecidos e utilizados CMMI e o MPS-br. No caso das entrevistadas, 41% dizem conhecer ou utilizar como referência o CMMI, 27% afirmam o mesmo sobre o MPS-br, 9% dizem seguir outras metodologias e 25% desconhece qualquer tipo de modelo de maturidade de processos (Figura 41).

Figura 41 - Modelos de Maturidade de Processos.



Fonte: Questionário aplicado nas empresas.

A implantação destas metodologias auxilia bastante da melhoria continua do processo de desenvolvimento, porém, são metodologias burocráticas, com alta quantidade de documentação, que exigem um custo considerável quanto às certificações, alem de que exigem muita disciplina e envolvimento de todos os funcionários da empresa.

Para a realidade do mercado de Passo Fundo não é uma tarefa fácil implementar tais metodologias, mas é um bom direcionamento para o futuro das organizações do município.

Um problema comum, que deve ser evitado, em grande quantidade de empresas, é o fato de buscarem estas certificações apenas com o intuito de obter status e visibilidade, não com o pensamento de melhorar seu processo e aumentar a qualidade de seus produtos em si. Sendo assim, muitas empresas até conseguem alcançar seus objetivos quanto a certificações de qualidade e maturidade, mas quando chegam ao topo deste processo de melhoria não tem capacidade ou não mantém seus esforços para continuar a melhoria. Isso explica o fato de muitas empresas com ISO-9001 decretarem falência.

4 CONSIDERAÇÕES FINAIS

Contudo o que foi apresentado neste trabalho através da associação do estudo bibliográfico com os resultados obtidos através da pesquisa foi possível identificar o perfil das empresas de Passo Fundo, determinar o que esta sendo bem gerido, foram observadas falhas e propostas soluções, além de identificar fatores potenciais das empresas que podem ser mais bem explorados.

As empresas de Passo Fundo, na sua maioria, estão com o direcionamento correto quanto a suas pretensões para o futuro, porém, precisam arriscar mais e tentar fazer com que a definição e melhoria continua de seu processo aconteçam na prática. Muitas delas gerenciam seu processo de desenvolvimento de uma maneira funcional, mecanicista e fechada, onde apenas os gerentes e analistas tomam as decisões.

Portanto, é necessário que os setores de TI estejam alinhados com os negócios da empresa, se desfazendo da idéia de que quem desenvolve precisa apenas saber programar e ser restritamente técnico, passando para um modelo de gerenciamento aberto a opiniões e discussões de maneira democrática e sempre respeitando as hierarquias para que se mantenha a ordem e a liderança. Sendo assim, existem momentos em que é preciso parar para planejar, para ganhar fôlego e pensar na melhoria em conjunto, com toda equipe de desenvolvimento, gerentes, analistas e envolvidos nos projetos. Diminui-se quantidade de novas demandas, para posteriormente ganhar em qualidade.

Além disso, os gerentes de projeto, analistas e desenvolvedores precisam voltar seus olhos para o mercado e para seus clientes, ser mais críticos quanto aos softwares que estão desenvolvendo e procurar *feedback* mais constante, para que desta forma promovam de fato a melhoria continuada de seus processos, da qualidade de seus produtos e obtenham a satisfação garantida de seus clientes. Afinal, quem desenvolve um software sempre vai defender todos os argumentos possíveis, mas acaba operando e testando ele de uma maneira que erros comuns passam despercebidos, por este motivo é importante ouvir opiniões e estar atento a oportunidades.

Planejamento, padronização, organização e disciplina no decorrer do desenvolvimento sempre resultarão em softwares com nível de manutenibilidade maior após sua entrega.

Os objetivos definidos para este trabalho foram alcançados e obtidos resultados satisfatórios que serão úteis para projetos futuros voltados ao mercado de software de Passo Fundo.

A metodologia utilizada respondeu bem as necessidades do projeto, porém, foram encontradas dificuldades para definição das perguntas, dos objetivos e das variáveis que precisavam ser criadas no questionário. Isso aconteceu pelo fato de a definição destes fatores estarem acontecendo paralelamente ao estudo bibliográfico e por ainda não ter realizada nenhum tipo de pesquisa junto às empresas. Por exemplo, se o questionário fosse criado após a revisão bibliográfica ou se tivesse sido criado um pré-questionário, seria bem mais fácil de estabelecer os parâmetros e informações que precisavam ser coletadas, mas isso é natural, devido ao ganho gradativo de conhecimento e experiência ao decorrer do desenvolvimento do projeto.

Outra dificuldade encontrada, que influenciou no citado anteriormente, foi a delimitação do tema que começou de maneira muito aberta e com o decorrer dos estudos foi ganhando direcionamento e foco.

A aplicação do questionário junto as empresas foi complicado pelo fato destas estarem sempre em produção e também por receio de abrir informações sobre seu processo, práticas de desenvolvimento e estratégias de negócios. Apesar de tudo isso, a quantidade de questionários respondidos superou expectativas, com um total de 16 entrevistados, número considerado satisfatório, considerando o número de empresas de desenvolvimento de Passo Fundo que tinham condições de somar informações relevantes ao projeto.

Como trabalhos futuros podem ser citados: a continuidade desta pesquisa, reformulando o questionário para atender a novos objetivos, aumentando o número de entrevistados, aplicação de questionário junto aos clientes das empresas entrevistadas para obter sua visão quanto a qualidade dos softwares entregues; criação de um software de *Help Desk* que atenda as necessidades das empresas estudadas, com base nos resultados obtidos; entre outros.

Sendo assim, para poder melhorar estes resultados em futuras análises, basta que as propostas de soluções transformem-se em ações e aconteçam nos ambientes de trabalho das empresas de desenvolvimento de Passo Fundo.

REFERÊNCIAS

ARAÚJO, Marcos; SOUZA, Vinicius; VALE, Ricardo. O papel evolutivo do software. Revista de Engenharia de Software, Edição 28, 2011. Disponível em: <http://www.devmedia.com.br/websys.5/webreader.asp?cat=48&artigo=2853&revista=esmagazine_28#a-2853>. Acesso em: 10 de maio de 2013.

BERTHOLDO, Leonardo; BARBAN, Lídia Regina de Carvalho Freitas. Adaptação do Scrum ao Modelo Incremental. Universidade Estadual de Campinas, Faculdade de Tecnologia, Limeira, São Paulo, 2010. Disponível em: <http://www.ft.unicamp.br/liag/Gerenciamento/monografias/Monografia_ModeloIncremental_SCRUM.pdf>. Acesso em: 31 de março de 2013.

BISSI, Wilson. Scrum: Metodologia de Desenvolvimento Ágil. Centro Universitário de Maringá (CESUMAR), Tecnologia em Análise e Desenvolvimento de Sistemas, Maringá, Paraná, 2007. Disponível em: <http://revista.grupointegrado.br/revista/index.php/campodigital/article/viewFile/312/146>. Acesso em: 19 de abril de 2013.

CAMPELO, Renata. XP-CMM2: Um Guia para Utilização de Extreme Programming em um Ambiente Nível 2 do CMM. Universidade Federal de Pernambuco, Centro de Informática, Recife, Pernambuco, 2003. Disponível em: <http://www.di.ufpe.br/hermano/download/dissertacoes/XP-CMM2.pdf>. Acesso em: 18 de abril de 2013.

CORDEIRO, Edson dos Santos; Introdução a Ciclo de Vida do Software, 2006. Disponível em: <http://www.cordeiro.pro.br/aulas/engenharia/processoDeSoftware/ciclos.pdf>. Acesso em: 20 de junho de. 2012.

DAMASCENO, Cristiane S.; ARAÚJO, Tiago V.; NUNES, Claudio. ITIL: Uma avaliação sobre as melhores práticas e os resultados de sua empregabilidade para corporações de porte variados, Departamento de Pós-Graduação, Universidade de Santa Cecília, Santos, São Paulo, 2009. Disponível em: < http://sites.unisanta.br/revistaceciliana/edicao_01/1-2009-43-56.pdf>. Acesso em: 28 de junho de 2013.

DIEHL, Astor Antonio; TATIM, Denise Carvalho. Pesquisa em ciências sociais aplicadas. São Paulo, 2004.

FALBO, Ricardo de Almeida. Engenharia de Software – Notas de Aula, Universidade Federal do Espírito Santo, 2005. Disponível em: <http://www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/NotasDeAula.pdf>. Acessado em: 09 de abril de 2012.

GROFFE, Renato. Gerenciamento de serviços de TI com ITIL: uma visão geral, Artigos Dev Media, 2012. Disponível em: < http://www.devmedia.com.br/gerenciamento-de-servicos-de-ti-com-itil-uma-visao-geral/25230>. Acesso em: 29 de junho de 2013.

GOOGLE, Images. Repositório digital de imagens da Google Disponível em: <https://www.google.com.br/imghp?hl=pt-BR&tab=wi>. Acesso em: 20 de junho de 2012.

McDONALD, Andrew; WELLAND, Ray. The University, Glasgow G12 8QQ. Scotland. 2001. Disponível em: <http://www.dcs.gla.ac.uk/~andrew/webe2001>. Acesso em: 20 de junho de 2012.

NASCIMENTO, Rafael Vasconcelos. Evolução de Software. Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO), Certificação Digital Nº 0210500/CB, Rio de Janeiro, 2010. Disponível em: < http://pt.scribd.com/doc/37217597/Evolucao-de-Software>. Acesso em: 06 de maio de 2013.

PFLEEGER, S. L. Engenharia de Software: Teoria e Prática, Prentice Hall do Brasil, 2ª Edição, 2004.

PINHEIRO, Flávio. Fundamentos no Gerenciamento de Serviços de TI com base na ITIL V3, TI Exames, São Paulo, São Paulo, 2011. Disponível em: <http://tiexames.com.br/Amostra_Apostila_ITIL_V3_Foundation.pdf>. Acesso em: 28 de junho de 2013.

PORTELLA, Cristiano. Temas de Aula: Manutenção de Software. Pontifícia Universidade Católica de Campinas (PUC-Campinas), Campinas, São Paulo, 2010. Disponível em: < http://www.cesarkallas.net/arquivos/faculdade/engenharia_de_software/16-Manuten%87_o%20de%20software/Manuten%87_o%20de%20Software.pdf>. Acesso em: 27 de maio de 2013.

PRESSMAN, R. S, Engenharia de Software, 6ª Edição, AMGH Editora, 2010.

RABELLO, Márcia R.; BORTOLI, Ray. Estrela: um processo de desenvolvimento para aplicações de comércio. Universidade de Passo Fundo, UPF Editora, Passo Fundo, Rio Grande do Sul, 2006.

ROESCH, Sylvia Maria. Projetos de estágio do curso de administração: guia para pesquisas, projetos, estágios e trabalho de conclusão de curso. São Paulo, 1996.

SCHWABER, Ken. Guia do Scrum. Scrum Aliance, 2009. Disponível em: <http://www.trainning.com.br/download/GUIA_DO_SCRUM.pdf >. Acesso em: 18 de abril de 2013.

SOFTWARE ENGINEERING INSTITUTE. The capability maturity model: guides for improving the software process. Reading: Addison Wesley, 1997.

SISNEMA, Informática. Fundamentos em ITIL, Versão 2.9, Porto Alegre, Rio Grande do Sul, 2009.

SOMMERVILLE, Ian, Engenharia de Software, 9th Edition, São Paulo: Pearson Prentice Hall, 2011.

WILDT, Daniel; LACERDA, Guilherme. Conhecendo o Extreme Programming (XP). Coletânea dos Trabalhos de CMP, Engenharia de Software, PPGC-UFRGS, 2011. Disponível em: <http://www.slideshare.net/dwildt/conhecendo-o-extreme-programming>. Acesso em: 08 de abril de 2013.

ANEXOS E APÊNDICES

ANEXO I – QUESTIONÁRIO APLICADO NAS EMPRESAS DE DESENVOLVIMENTO DE PASSO FUNDO

1) A quanto tempo a empresa está no mercado?

_____

_____

2) Quantos funcionários possui?

_____

_____

3) A empresa trabalha com:

- Desenvolvimento Web;
- Desenvolvimento Desktop;
- Desenvolvimento Mobile;
- Outro.

4) Linguagens de programação utilizadas pela empresa?

- PHP;
- JAVA;
- Ruby;
- Python;
- Visual Basic;
- Linguagem C;
- CSS;
- HTML;
- Android;
- PHP Mobile;

- Outro.

5) Existe divisão de setores? Exemplo: Manutenção, Desenvolvimento, Vendas, Financeiro, etc.

- Sim;
- Não.

6) Se a resposta anterior foi "Sim", descreva brevemente como é esta divisão.

_____

_____

7) Quantos clientes atende?

_____

_____

8) Quais os principais cuidados que a empresa toma para desenvolver um produto que atenda aos objetivos (necessidades) de seus clientes?

_____

_____

9) Sua definição sobre processo de software?

- Extremamente Clara;
- Parcialmente Clara;
- Pouco Clara;
- Desconheço o termo.

10) Se possível, descreva o que você entende sobre processo de software.

_____

_____

11) Modelos de processo utilizados na empresa:

- Modelo Cascata;
- Modelo Incremental;
- Modelo de Prototipagem;
- Modelo RUP;
- XP (Extreme Programming);
- FDD (Desenvolvimento Guiado por Características);
- SCRUM;
- Não utilizamos;
- Não temos conhecimento;
- Outro.

12) Levando em consideração a participação e o envolvimento de sua equipe, incluindo você, como você classifica a utilização da metodologia de desenvolvimento da empresa:

- O processo é totalmente executado;
- O processo é parcialmente executado;
- O processo não é executado;
- Não existe processo.

13) Em sua opinião, o que é mais relevante no processo de desenvolvimento para termos um produto (software) de qualidade?

_____

_____

14) É realizada documentação das etapas do processo de desenvolvimento?

- Sim;
- Não.

15) Utiliza-se de alguma ferramenta ou software de gerenciamento ou apoio ao processo de desenvolvimento de software?

- Sim;
- Não.

16) Você trabalha a melhoria do seu processo de software?

- Sim;
- Não.

17) Se sua resposta anterior foi "Sim", descreva qual é o principal fator da busca pela melhoria em sua empresa.

_____

_____

18) Já ouviu falar ou utiliza modelos de maturidade de processo de software?

- CMMI;
- MPS-br;
- Desconheço os modelos;
- Outro.

19) Caso utilize algum dos modelos citados anteriormente, em que nível esta?

_____

_____

20) Sua empresa consegue associar bem os processos de apoio (suporte e manutenção) com os outros processos de desenvolvimento sem interferir no andamento das tarefas?

- Facilmente associado;
- Parcialmente associado;

- Temos dificuldades para associar;
- Não conseguimos associar e temos problemas com tempo no desenvolvimento de alguns projetos por esse motivo.

21) Sua empresa se preocupa com a evolução do software desenvolvido, no sentido de prover procedimentos que facilitem essa evolução do software após a entrega ao cliente?

- Sim;
- Não.

22) Utiliza-se de algum software que auxilie na gerência das mudanças, por exemplo, um gerenciador de versões?

- Sim;
- Não.

23) Existe algum setor ou alguém responsável na empresa, para apenas atender os clientes prestando manutenção e suporte, separadamente do desenvolvimento?

- Sim;
- Não.

24) Utiliza-se de software de Help Desk?

- Sim;
- Não.

25) Se utiliza software de Help Desk, qual software é esse?

_____

_____

26) Descreva brevemente como é organizado o suporte e manutenção aos clientes. Quais serão as etapas para o atendimento a um cliente?

_____

_____

27) Todos os projetos de sua empresa são desenvolvidos pensando no futuro do software, prevendo possíveis alterações posteriores?

- Todos;
- A maior parte dos projetos;
- Alguns projetos apenas;
- Nenhum dos projetos.

28) Sua empresa classifica ou divide as tarefas de suporte de acordo com o tipo de manutenção? Exemplo: Correção de Defeitos, Adaptação Ambiental, Adição de Funcionalidade.

- Sim;
- Não.

29) O que você acha que poderia melhorar em sua empresa no suporte ao cliente?

_____

_____

30) De uma nota de 1 a 5, em relação ao que você acredita quanto a satisfação de seus clientes.

_____

_____