PROJETO DE GRADUAÇÃO

ANÁLISE DAS ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE NO CENTRO DE PROCESSAMENTO DE DADOS DE UMA UNIVERSIDADE PÚBLICA

Por,

Mayllon Melo de Miranda

Brasília, novembro de 2016

UNIVERSIDADE DE BRASILIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE PRODUÇÃO

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Departamento de Engenharia de Produção

PROJETO DE GRADUAÇÃO

ANÁLISE DAS ATIVIDADES DE DESENVOLVIMENTO DE SOFTWARE NO CENTRO DE PROCESSAMENTO DE DADOS DE UMA UNIVERSIDADE PÚBLICA

POR,

Mayllon Melo de Miranda

Relatório submetido como requisito para obtenção

do grau de Engenheiro de Produção

Banca Examinadora

PhD Sanderson Cesar Macedo Barbalho – UnB / EPR (Orientador) _____

PhD Claudia de Oliveira Melo – UnB / EPR (Examinadora) _____

Brasília, novembro de 2016

RESUMO

Com a tendência de passagem do Brasil para o grupo de economias com maior grau de maturidade, que privilegiam o desenvolvimento de soluções e sistemas, enxergou-se a importância de compreender como esse processo acontece. Este estudo objetiva mapear as atividades de desenvolvimento de *software* em um órgão público de administração direta, de modo a verificar possíveis falhas que impeçam o sucesso total dos projetos de *software*. O trabalho foi realizado através de um estudo de caso, no qual o processo de desenvolvimento de *software* foi modelado na sua forma vigente em um Centro de Informação de uma Instituição de Ensino Superior pública brasileira. A partir da análise do modelo, ele foi classificado de acordo com os modelos de desenvolvimento de *software*, posteriormente, foi confrontado quanto ao seu modo de gestão de projetos, e por fim foi comparado com metodologias ágeis de desenvolvimento de *software*.

Palavras-chave: desenvolvimento de *software*, órgão público, gestão de projetos, metodologias ágeis, mapeamento de processos.

ABSTRACT

With the tendency of Brazil to move to the group of economies with a higher degree of maturity, which favor the development of solutions and systems, it was seen the importance of understanding how this process happens. This study aims to map software development activities in a public direct administration agency, in order to verify possible failures that prevent the total success of software projects. The work was carried out through a case study, in which the software development process was modeled in its current form in an Information Center of a Brazilian Public Higher Education Institution. From the analysis of the model, it was classified according to the models of software development, later, it was confronted as to its mode of project management, and finally it was compared with agile methodologies of software development.

Keywords: software development, public agency, project management, agile methodologies, process mapping.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

1. INTRODUÇÃO

1.1 ASPECTOS GERAIS

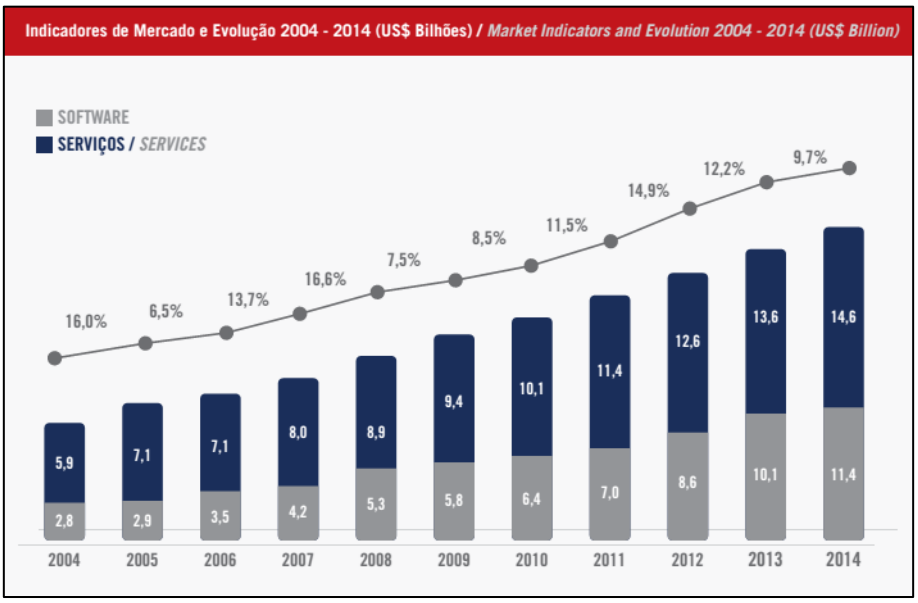
Desde 2005, a Associação Brasileira das Empresas de *Software* – ABES e a *International Data Corporation* – IDC, importante empresa de consultoria especializada em TI (tecnologia da informação), telecomunicações e inteligência de mercado de TI, realizam e divulgam o Estudo “Mercado Brasileiro de *Software* – Panorama e Tendências”, que permite conhecer e analisar as oportunidades de tecnologia deste segmento econômico. Ele exibe, detalhadamente, dados seguros sobre os mercados brasileiro e mundial de *software*: taxas de crescimento, evolução e tendências do setor; informações fundamentais para o direcionamento das estratégias empresariais.

De acordo com a pesquisa divulgada em 2015, a Indústria Brasileira de TI possui o 7° maior investimento mundial, com US$ 60 bilhões desembolsados no ano de 2014, estando atrás apenas de Estados Unidos, China, Japão, Reino Unido, Alemanha e França, respectivamente. Contudo, a taxa de crescimento do investimento em TI no Brasil vem arrefecendo em relação aos anos anteriores, apesar de assinalar um aumento de 6,7% em relação a 2013, que ainda é superior à média mundial que registrou crescimento de 4,04%. O Estudo exibe também que o Brasil está posicionado em 1º lugar no ranking de investimentos no setor de TI na América Latina, sendo responsável por 46% desse mercado que soma US$ 128 bilhões, número este, ainda tímido se comparado ao faturamento mundial, o qual atingiu a marca de US$ 2,090 bilhões, somando-se todos os mercados internos, isto é, excluindo as exportações.

O mercado doméstico de TI, que inclui hardware, *software* e serviços, representa 2,6% do PIB brasileiro e 3% do investimento total em TI no mundo. Separadamente, o faturamento do mercado de *software* do Brasil atingiu a marca de US$ 11,2 bilhões, sem exportações; já o mercado de serviços registrou valor na ordem de US$ 14 bilhões, enquanto que o mercado de hardware acumulou US$ 34,8 bilhões em 2014. A produção local de programas de computador cresceu 19,1% e alcançou praticamente ¼ do total do mercado, enquanto que a produção desenvolvida no exterior registrou índice de crescimento de apenas 11,5%. Das 12.660 empresas atuantes no setor de *software* e serviços identificadas em 2014, cerca de 55% delas operam no desenvolvimento e produção ou na prestação de serviços, enquanto que o restante se dedica à distribuição e comercialização. Juntos, *software* e serviços tiveram um crescimento de 9,7% de 2013 para 2014, acima da grande maioria dos demais setores da economia brasileira. Para corroborar com esses números, a Figura 1 mostra a evolução de mercado de *software* e serviços no Brasil, permitindo desta forma notar que a soma desses dois segmentos esteve superior ao PIB do país nos últimos 10 anos de divulgação do Estudo. Além disso, a soma destes segmentos continua a se manter

acima dos 40% do mercado total de TI, o que aponta para uma tendência de passagem do país para o grupo de economias com maior grau de maturidade, que privilegiam o desenvolvimento de soluções e sistemas.

Figura 1 – Evolução de Mercado de *Software* e Serviços no Brasil.



Fonte: ABES; IDC, 2015.

O mercado brasileiro de *Open Source* – termo em inglês que significa código aberto e diz respeito ao código-fonte de um *software*; representa cerca de 5% de todo o mercado de *softwares* e serviços. Destes, quase 63% dos investimentos são realizados pelo Governo, ou seja, US$ 765 milhões. Essa expressividade em participação no mercado de código aberto por parte da esfera pública se deve ao seguinte fato:

> O *software* livre é uma opção estratégica do Governo Federal para reduzir custos, ampliar a concorrência, gerar empregos e desenvolver o conhecimento e a inteligência do país na área. Para incentivar o uso do *software* livre, o Estado promove ações voltadas para o uso de padrões abertos, o licenciamento livre dos *softwares* e a formação de comunidades interessadas no tema (GOVERNO ELETRÔNICO).

O Serviço Federal de Processamento de Dados (Serpro) é uma empresa pública vinculada ao Ministério da Fazenda e tem como objetivo, modernizar e dar agilidade a setores estratégicos da Administração Pública brasileira. Para isso, o Serpro desenvolve programas e serviços que permitem maior controle e transparência sobre a receita e os gastos públicos, de modo a investir no desenvolvimento de soluções tecnológicas em *Software* Livre. Seu orçamento é composto pelo Ministério da Fazenda, que corresponde a 65% do volume de negócios da empresa, e os outros 45% pelo Ministério do Planejamento, Orçamento e Gestão, de tal forma que somaram R$ 1,9 bilhões no ano de 2015 (SERPRO). Portanto, fica claro que o serviço público federal adquire muito *software*, tem organismos dedicados ao desenvolvimento de *software*, e ainda sim, desenvolve internamente.

1.1.1 GERENCIAMENTO DE PROJETOS DE SOFTWARE NO BRASIL

O Professor Darci Prado lançou em dezembro de 2002 o Modelo de Maturidade de Gerenciamento de Projetos (modelo Prado-MMGP), o qual se propõe a mensurar o estágio de uma organização na habilidade de gerenciar seus projetos com sucesso, ou seja, avalia a maturidade da organização.

O modelo Prado-MMGP aborda todo o ciclo de criação do bem (produto ou serviço) envolvendo desde processos finalísticos a processos de suporte; reflete o uso de boas práticas de gestão, principalmente aquelas que realmente agregam valor; relaciona a maturidade da organização com sua capacidade de executar projetos com sucesso; utiliza os mesmos níveis do modelo SW-CMM (1 até 5) desenvolvido pela *Carnegie-Mellon University* para desenvolvimento de *software*; e considera características como simplicidade (questionário com 40 questões) e universalidade (ser aplicável a todo tipo de organização e a toda categoria de projeto).

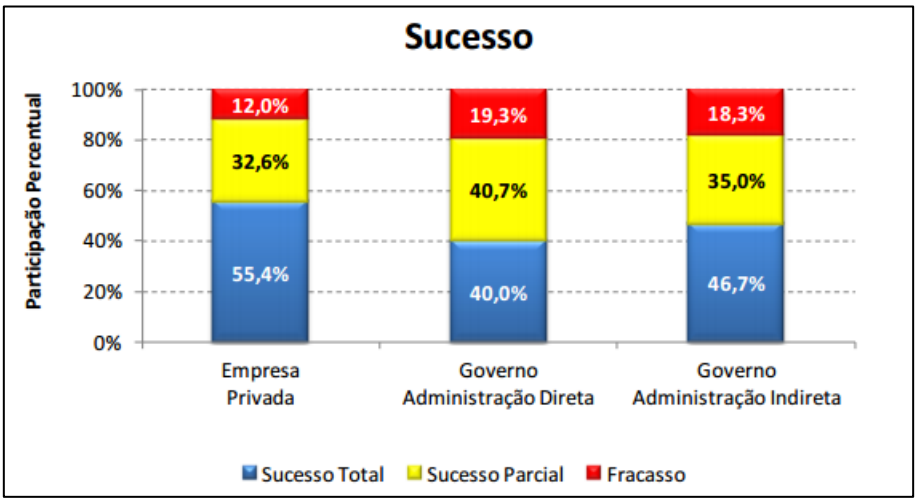
Desde 2005, Archibald (um dos fundadores do *Project Management Institute* – PMI nos Estados Unidos) e Prado divulgam a diversas entidades e formadores de opinião a Pesquisa Maturidade em Gerenciamento de Projetos, que consiste num relatório geral subdividido em várias áreas de atuação dos mais diversos setores econômicos. Assim, ao analisar a composição da carteira de projetos de Desenvolvimento de Novos Aplicativos (*software*) – lembrando que há outras categorias de projeto analisadas pela pesquisa, o estudo indicou que cada organização desenvolve em média 23 projetos simultaneamente, que duram em média 8 meses cada um, ao valor de investimento médio de R$ 11.165.000,00. A maturidade global média observada no ano de 2012 é de 2,68 com uma significativa presença de organizações no nível 3-padronizado (37,5%). Para 50,0% (níveis 1-inicial e 2-conhecido) das organizações, o gerenciamento de projetos ainda não possibilita trazer resultados aos seus negócios tal como seria desejado. E somente 12,3% das organizações estão situadas em patamares que permitem o domínio e a otimização do trabalho (níveis 4-gerenciado e 5-otimizado).

Em termos de maturidade por tipo de organização, a pesquisa mostrou que as empresas de iniciativa privada e do governo (administração indireta) obtiveram o mesmo resultado: 2,72. Enquanto que as organizações públicas de administração direta foram classificadas com nível de 2,14. A Figura 2, a Figura 3 e a Figura 4 reforçam a iniciativa da pesquisa em comparar esses tipos de organização separadamente.

A Figura 2 refere-se ao sucesso do desenvolvimento dos projetos de *software*, na qual avalia as organizações em termos de entrega do produto/serviço de acordo com a seguinte classificação:

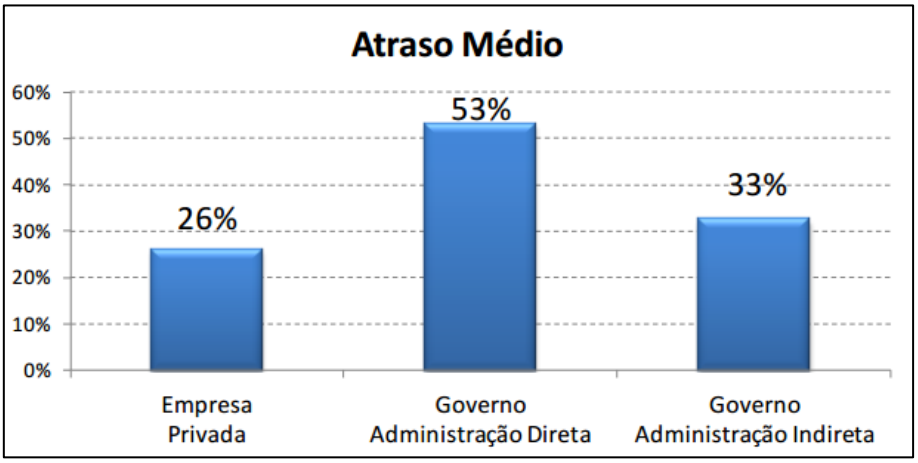
- Sucesso total: o projeto encerrou no prazo, o escopo e o orçamento previstos foram respeitados, o usuário ficou totalmente satisfeito, pois o produto/serviço que lhe foi entregue está sendo utilizado e realmente agregou valor ao seu trabalho;
- Sucesso parcial: o projeto foi encerrado e o *software* está sendo utilizado, no entanto, houve atraso e/ou estouro de orçamento significativo. Além de o produto/serviço não apresentar todas as funcionalidades esperadas e/ou não agregar o valor esperado ao seu trabalho;
- Fracasso: o projeto foi paralisado ou o produto/serviço entregue não está sendo utilizado por não atender às expectativas dos usuários ou o atraso foi tal que implicou em perdas para o negócio. O usuário/cliente ficou profundamente insatisfeito.

Figura 2 – Sucesso no Desenvolvimento de Projetos de *Software*.



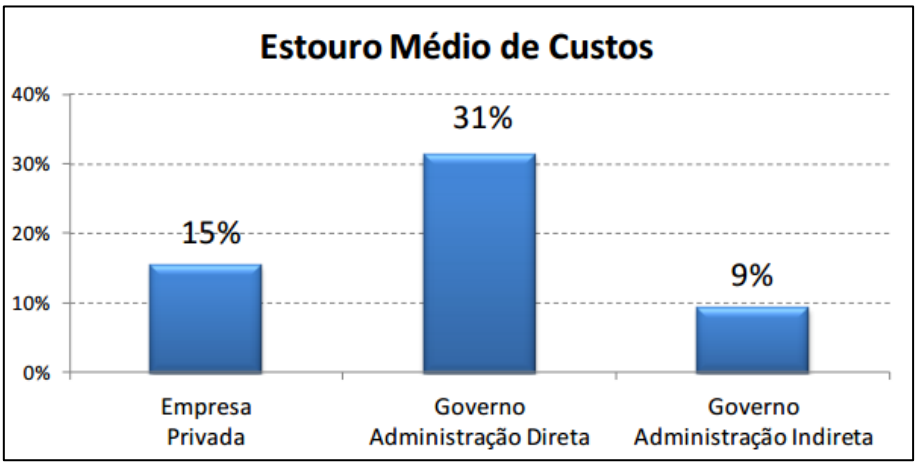
Fonte: ARCHIBALD; PRADO (2012).

Figura 3 – Atraso Médio no Desenvolvimento de Projetos de *Software*.



Fonte: ARCHIBALD; PRADO, 2012.

Figura 4 – Estouro Médio de Custos no Desenvolvimento de Projetos de *Software*.



Fonte: ARCHIBALD; PRADO, 2012.

A Figuras 3 e a Figura 4 mostram que o atraso e o estouro do orçamento são dificuldades recorrentes nos projetos de desenvolvimento de *software* das organizações públicas de administração direta e isto reflete diretamente no alto grau de fracasso (19,3%) apresentado pela Figura 1. Como consequência, é possível inferir que existe uma relação positiva entre os níveis de maturidade e o sucesso total, assim como com a soma de sucesso total e sucesso parcial. Por outro lado, existe uma relação invertida entre os níveis de maturidade e fracasso, bem como com o atraso médio e com o estouro médio de custos.

Segundo Carvalho e Mello (2012), a aplicação do método ágil SCRUM (método que divide o desenvolvimento em diversas iterações de ciclos mais curtos e realizam entregas ao final de cada uma delas, de forma que o cliente interno e/ou externo receba uma versão que agregue valor ao seu negócio) em desenvolvimento de produtos de *software* traz a organização seis benefícios: (1) melhoria na comunicação e aumento da colaboração entre envolvidos; (2) aumento da motivação da equipe de desenvolvimento; (3) diminuição no tempo gasto para terminar o projeto; (4) diminuição do risco do projeto – menor possibilidade de insucesso; (5) diminuição dos custos de produção (mão de obra); e (6) aumento de produtividade da equipe. Contudo, não foi possível verificar se houve aumento da qualidade do produto, da satisfação dos clientes e do retorno do investimento do projeto.

1.1.2 FORMULAÇÃO DO PROBLEMA

Gil (2002) considera o problema como sendo uma questão não solvida e que é objeto de discussão em qualquer domínio do conhecimento. Porém, nem todo problema é passível de tratamento científico, portanto, primeiramente é preciso averiguar se o problema pertence a categoria de científico. O autor indica que para ser considerado científico, o problema deve ser proposto de forma a possibilitar a investigação segundo os métodos próprios da ciência, isto é, a formulação de problemas científicos exige que ele seja formulado como pergunta,

seja claro e preciso, seja empírico ou teórico, seja suscetível de solução; e que seja delimitado a uma dimensão viável.

A proposta em tela se concentra nas técnicas de gestão necessárias para planejar, organizar, monitorar e controlar projetos de *software*. Desse modo, o presente trabalho pretende responder à seguinte questão problemática observada pelo Centro de Processamento de Dados (CPD) da Instituição de Ensino Superior (IES) pública apontada como caso deste objeto de estudo:

- O CPD desta IES pública realiza seus projetos de *software* de uma forma padronizada, eficiente, com resultados efetivos e sucesso?

Uma vez respondida, positivamente, essa questão, o gerenciamento de projetos de *software* da universidade pública estará apto a entregar um produto/serviço com sucesso total (maior qualidade, dentro do prazo, escopo e orçamento atendidos, usuário satisfeito e valor agregado), ou seja, estará contribuindo para o fluxo de trabalho dos docentes, a formação dos discentes e até, indiretamente, para o progresso da comunidade que a envolve.

1.2 OBJETIVOS

Objetivo Geral:

Analisar, a partir de mapeamentos de processos, as atividades de desenvolvimento de *software* em um órgão público de modo a identificar possíveis deficiências que impeçam o sucesso total dos projetos de *software*.

Objetivos Específicos:

- Entender a metodologia tradicional de desenvolvimento de *softwares*;
- Diagnosticar o processo de desenvolvimento de *software* da instituição pesquisada;
- Comparar as diferenças entre as metodologias (tradicional e observada); e
- Correlacionar as deficiências encontradas com os parâmetros de insucesso de projetos de *software*.

1.3 JUSTIFICATIVA

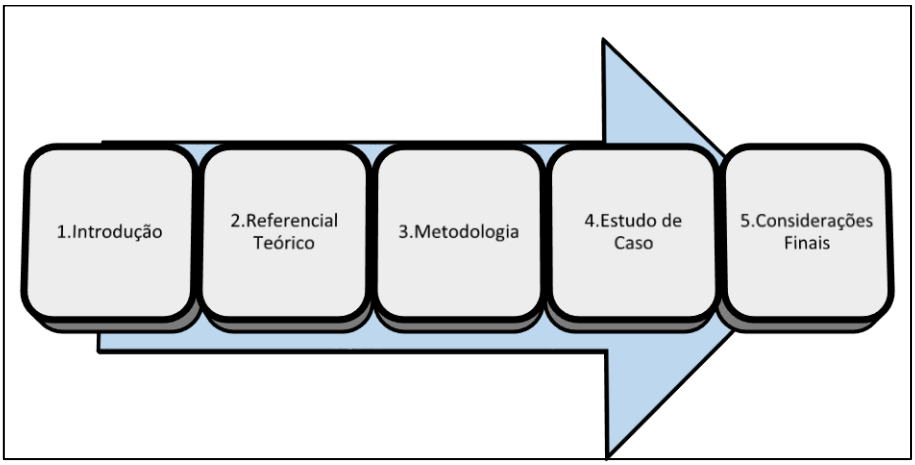
Pesquisar sobre a relação existente entre as atividades de desenvolvimento de *software* e o sucesso de projetos de *software* possui especial relevância para o conhecimento científico, pois esta pesquisa visa propiciar ao serviço público um melhoramento de sua eficiência através da otimização das atividades de desenvolvimento de *software*, assim, irá contribuir para o sucesso dos projetos de modo a confirmar ou até preencher lacunas no âmbito teórico. Com a gestão efetiva de projetos de *software*, o cliente será atendido de acordo com suas necessidades específicas (tempo e qualidade); a equipe do projeto terá seu trabalho reconhecido, e possivelmente, seus salários valorizados; a organização desenvolvedora do

software terá seus custos reduzidos e, consequentemente, seus lucros aumentados; em virtude do aumento da demanda por seus serviços, também necessitará de novos postos de trabalho, o que será benéfico ao país, aumentando sua arrecadação através de divisas tributárias.

1.4 ESTRUTURA DO TRABALHO

A Figura 5 apresenta a divisão dos cinco capítulos previstos nesse trabalho.

Figura 5 – Estrutura do Trabalho.



Fonte: o autor.

O Capítulo I, ora apresentado, introduz o projeto de pesquisa, sendo dividido em aspectos gerais do tema, formulação do problema, objetivos (geral e específicos), justificativa e estrutura do trabalho.

O Capítulo II tem como finalidade levantar o referencial teórico, isto é, descrever os fundamentos e conceitos mais gerais que embasam a pesquisa.

O Capítulo III visa a metodologia do trabalho, contemplando a classificação do tipo de pesquisa utilizado pelo trabalho e a estrutura em si proposta para o seu desenvolvimento.

O Capítulo IV exibe a consolidação os dados obtidos com a pesquisa, bem como a análise e os resultados da observação.

O Capítulo V, finalmente, apresenta as conclusões do trabalho e indica possibilidades de continuação da pesquisa.

2. REFERENCIAL TEÓRICO

2.1 TECNOLOGIA DA INFORMAÇÃO

Foina (2001) descreve a Informação como um valor ou um dado que tem utilidade para alguma aplicação ou pessoa. Deste modo, a Tecnologia da Informação (TI) é definida como um conjunto de métodos e ferramentas, mecanizadas ou não, que tem o objetivo de garantir

a qualidade e pontualidade dessas informações dentro da malha organizacional. Para atingir seus objetivos, a TI deve agir sobre os seguintes pontos:

- Definir conceitualmente os termos e vocábulos usados na organização;
- Estabelecer o conjunto de informações estratégicas;
- Atribuir responsabilidades pelas informações;
- Identificar, otimizar e manter o fluxo de informações corporativas;
- Mecanizar os processos manuais;
- Organizar o fluxo de informações para apoio às decisões gerenciais.

Grembergen e Haes (2009, apud RUSCH; OLIVEIRA, 2012) apresentam indicadores para a TI sob diversas perspectivas, os quais fazem referência à avaliação da percepção dos diversos envolvidos sobre os produtos e serviços de TI, ou seja, como são vistos os resultados de investimentos nos departamentos de TI. Os objetivos e indicadores mais usualmente recomendados como boas práticas para analisar o desempenho segundo essas perspectivas estão exibidos no anexo A, onde a primeira coluna do quadro apresenta as perspectivas de TI, enquanto que a segunda coluna traz os objetivos da gestão de portfólio de projetos de *software* em relação à contribuição da TI para o negócio. Esses objetivos representam descrição de alto nível para traduzir os indicadores mensuráveis apresentados na terceira coluna. O uso continuado dos indicadores propostos nessa perspectiva permite mensurar e otimizar o valor agregado por investimentos em projetos de TI e quantificá-los para as organizações (KEYES, 2005 apud RUSCH; OLIVEIRA, 2012).

A evolução histórica da TI nas organizações se inicia com o surgimento dos Centros de Processamento de Dados (CPD), Centros de Informação (CI), Centros de Suporte ao Usuário (CSU) até a eclosão dos Centros de Desenvolvimento de Sistemas (CDS), que por sua vez, são compostos por equipes de analistas e programadores internos capazes de desenvolver sistemas complexos, responsáveis pela disseminação da informática organizacional. O processo produtivo de sistemas de informação que ocorre dentro desses Centros é denominado engenharia de *software* (FOINA, 2001; PRESSMAN, 2006).

2.1.1 ENGENHARIA DE SOFTWARE

Bauer (1969 apud PRESSMAN, 2006) define engenharia de *software* como “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”. Enquanto que, para o *Institute of Electrical and Electronics Engineers* – IEEE (1993 apud PRESSMAN, 2006), a engenharia de *software* consiste no emprego de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção do *software*. Diante disso, Pressman (2006) caracteriza a engenharia de *software* como sendo uma tecnologia em camadas que integra processo, métodos e ferramentas para o desenvolvimento de

softwares (programas, dados e documentos) com a finalidade de fornecer uma estrutura para a construção de *software* com alta qualidade.

Figura 6 – Engenharia de *Software* em Camadas.

Fonte: PRESSMAN, 2006, p. 17.

Toda organização deve se apoiar num compromisso com a qualidade, buscando um processo contínuo de aperfeiçoamento. Por isso a base na qual a engenharia de *software* se ampara é o foco na qualidade. Os processos de *software* constituem o alicerce para o controle gerencial de projetos de *software* e determinam o cenário no qual os métodos são aplicados, os produtos de trabalho são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são geridas. Os métodos providenciam a técnica para construir os *softwares* através de um conjunto de tarefas que incluem comunicação, análise de requisitos, modelagem de projeto, construção de programas, testes e manutenção. E as ferramentas fornecem ajuda automatizada ou semiautomatizada para o processo e para os métodos de modo que, quando integradas, formam um sistema de apoio ao desenvolvimento de *software* no qual a informação criada por uma ferramenta é usada por outra (PRESSMAN, 2006).

2.1.2 MODELOS DE PROCESSO DE *SOFTWARE*

Para Pressman (2006), um modelo de processo de *software* é uma representação abstrata de um processo de *software*, que se caracteriza por um conjunto distinto de atividades, ações, tarefas e marcos que levam à produção de um produto de *software*. Este conjunto de elementos é aplicável à maioria dos projetos que são necessários para fazer engenharia de *software* com alta qualidade e é formado por:

- Comunicação: realiza o levantamento de requisitos em colaboração com o cliente;
- Planejamento: descreve as tarefas, os riscos, os recursos, os produtos e o cronograma do projeto;
- Modelagem: concebe os modelos que permitem ao desenvolvedor e ao cliente, entender melhor o projeto e seus requisitos;
- Construção: conduz a geração de código e os testes para identificação de erros;
- Implantação: entrega o *software* ao cliente, que avalia o produto e retorna com *feedback*.

Como visto acima, os modelos de processo de *software* iniciam-se com os requisitos. Assim sento, Kothe, Marx e Frank (2015) sugerem que os requisitos sejam amparados por cinco etapas principais: elicitação ou levantamento de requisitos, especificação e priorização de requisitos, gerenciamento do escopo da solução, validação e monitoramento; pois assim garantem a qualidade no processo de gestão dos requisitos do produto de *software*.

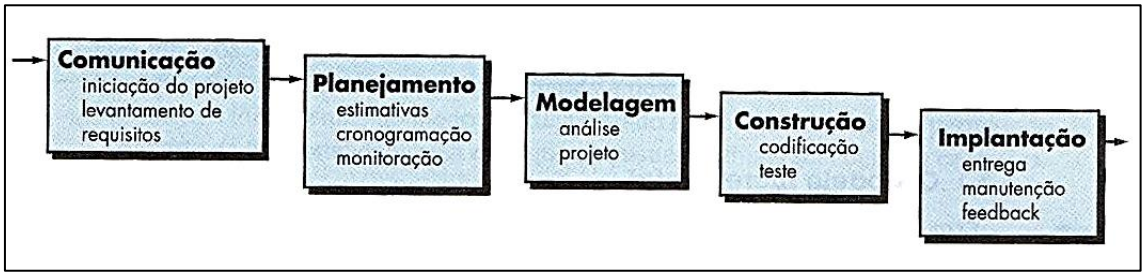
Os modelos tradicionais tratam dos elementos expostos acima de modo a organizá-los num fluxo de trabalho que mostre suas inter-relações, são eles: Cascata, Incremental e *Rapid Application Development* (RAD). Já os modelos evolucionários são mais iterativos, pois permitem desenvolver versões mais completas do *software*, são eles: Prototipagem, Espiral e Desenvolvimento Concorrente. Há ainda os modelos especializados, os quais são aplicados quando se deseja atingir uma meta específica de desenvolvimento de *software*, são eles: Desenvolvimento Baseado em Componentes, Métodos Formais e Desenvolvimento de *Software* Orientado a Aspectos.

2.1.2.1 Modelos Tradicionais

O modelo em cascata segue uma abordagem sistemática e sequencial para o desenvolvimento de *software*. Em princípio, o resultado de cada fase é consolidado em um ou mais documentos aprovados. Isto quer dizer que a fase seguinte não pode começar sem que a anterior tenha terminado. Por esse motivo, o modelo em cascata é adequado quando os requisitos são bem compreendidos, como em projetos de aperfeiçoamentos de sistemas existentes (PRESSMAN, 2006).

O maior problema do modelo em cascata é a divisão inflexível do projeto em estágios distintos. Devido a isso, é difícil estabelecer todos os requisitos inicialmente e somente na fase final do processo o *software* será colocado em uso numa versão executável para o cliente. Nesse momento, surgem novas necessidades de funcionalidades devido aos erros e omissões nos requisitos originais do *software*, o que envolve um retrabalho significativo e onera expressivamente o custo do projeto. Portanto, o modelo em cascata é inviável para a atualidade, pois ele não consegue acompanhar o ritmo rápido dos desenvolvimentos de *software* e as grandes torrentes de modificações de requisitos que os *softwares* sofrem (PRESSMAN, 2006).

Figura 7 – Modelo em Cascata.

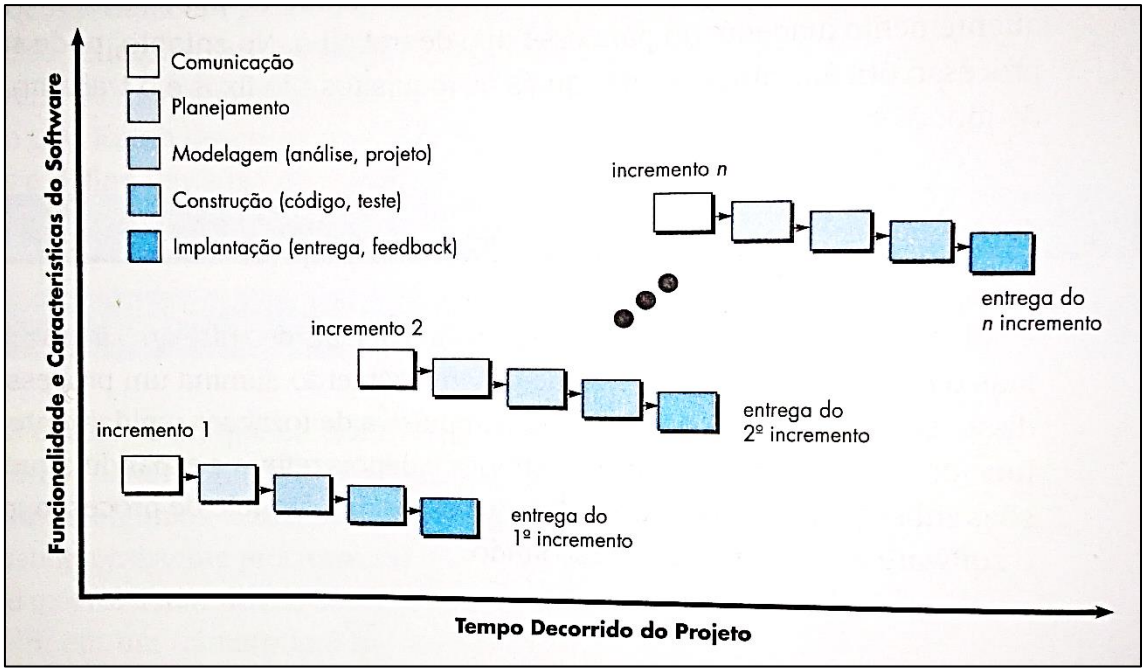


Fonte: PRESSMAN, 2006, p. 39.

O modelo incremental combina elementos do modelo em cascata aplicado de maneira iterativa, pois cada sequência produz incrementos do *software* passíveis de serem entregues, que fornecem progressivamente mais funcionalidade (MCDERMID, 1993 apud PRESSMAN, 2006). O modelo de processo incremental é particularmente útil quando não há recursos suficientemente disponíveis para a implementação completa. O primeiro incremento é chamado de núcleo do produto. Assim, se ele for bem-sucedido, um plano é desenvolvido para o próximo incremento como resultado do seu uso/avaliação. Portanto, existem uma série de vantagens (PRESSMAN, 2006):

- Os clientes não precisam esperar até a entrega do sistema completo para se beneficiarem dele;
- Os clientes podem usar incrementos iniciais como protótipos e ganhar experiência, obtendo informações sobre os incrementos posteriores;
- Existe um risco menor de falha geral do projeto.

Figura 8 – Modelo Incremental.



Fonte: PRESSMAN, 2006, p. 40.

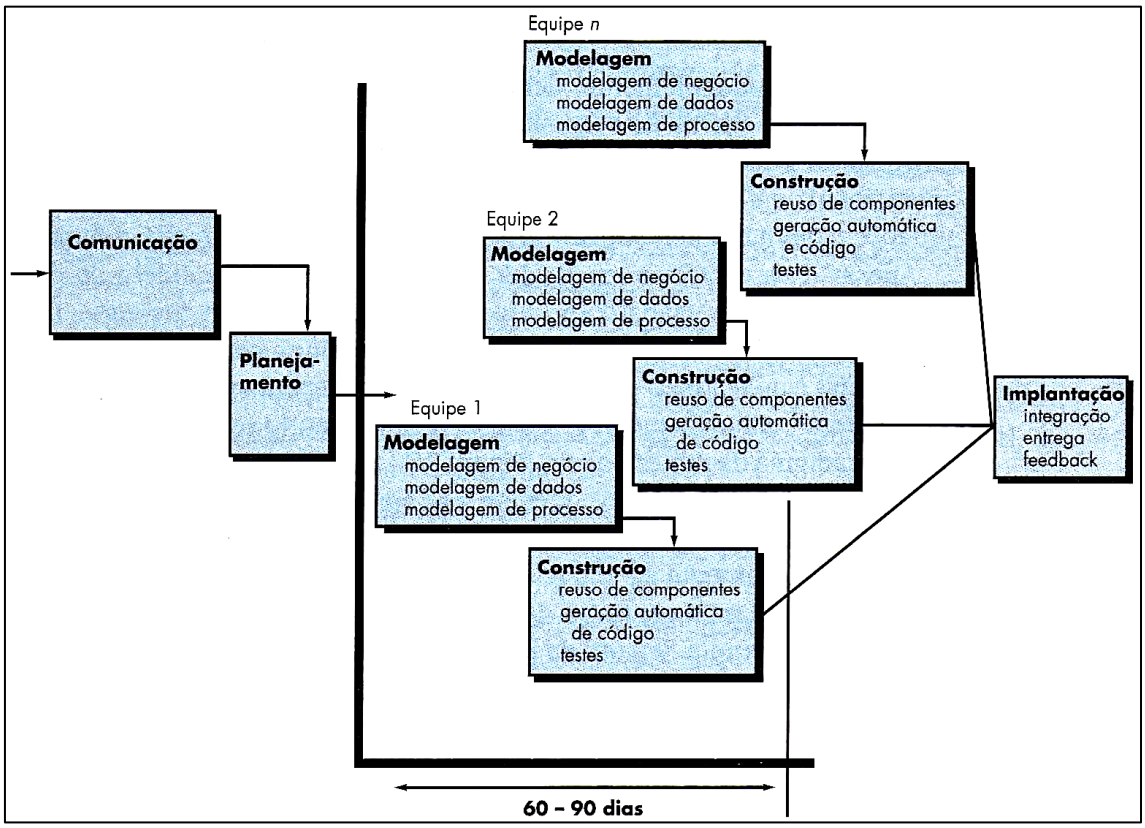
O modelo RAD enfatiza um ciclo de desenvolvimento curto com o uso de uma abordagem de construção baseada em componentes, ou seja, é uma adaptação de alta velocidade do modelo em cascata. A primeira etapa, comunicação, busca entender as características do negócio e do *software*. Em seguida, a etapa de planejamento prevê o trabalho em paralelo das várias equipes de *software* em diferentes funções do sistema. A modelagem é dividida em três fases: modelagem de negócios, modelagem dos dados e modelagem dos processos. A construção destaca o uso de componentes de *software* já existentes e a

aplicação de geração de códigos automática. Por fim, a implantação estabelece a base das iterações subsequentes (KERR, 1994 apud PRESSMAN, 2006).

O modelo RAD é recomendado para projetos de aplicação modularizada de modo que a função principal possa ser implementada em até três meses (MARTIN, 1991 apud PRESSMAN, 2006). Segundo Butler (1994 apud PRESSMAN, 2006) o modelo apresenta quatro desvantagens:

- Exige recursos humanos suficientes para criar um número adequado de equipes RAD em projetos grandes;
- Os desenvolvedores e os clientes precisam estar comprometidos com as atividades continuamente rápidas, para completar o sistema no curto espaço de tempo, senão o projeto RAD falhará;
- O sistema deve ser adequadamente modularizado para que as construções dos componentes não sejam comprometidas;
- Não é adequado quando é necessário um alto desempenho com muitas modificações dos componentes de sistemas ou quando os riscos técnicos são altos.

Figura 9 – Modelo RAD.



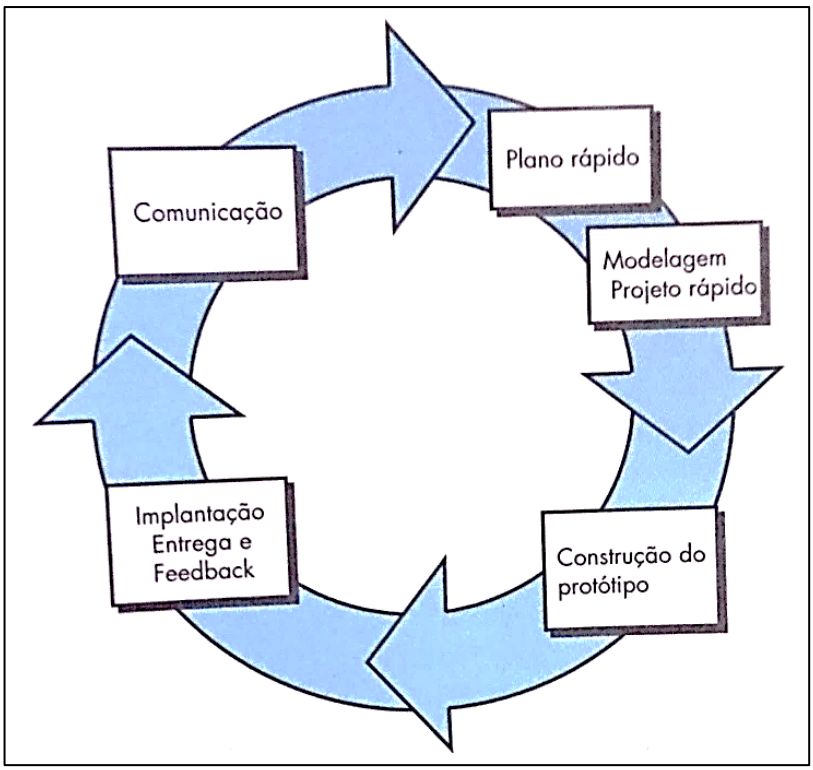
Fonte: PRESSMAN, 2006, p. 41.

2.1.2.2 Modelos Evolucionários

Os modelos evolucionários são explicitamente projetados para acomodar um produto que evolui com o tempo, pois a cada iteração, produzem uma versão cada vez mais completa do

software. O modelo de prototipagem, idealmente, serve como um mecanismo de identificação dos requisitos do *software*. Segundo Brooks (1975, apud PRESSMAN, 2006) tanto os clientes como os analistas gostam do modelo de prototipagem. Os clientes porque tem um sistema real nas mãos e os desenvolvedores porque conseguem construir algo imediatamente. No entanto, é preciso lembrar que o protótipo serve como um primeiro sistema, o qual será descartado posteriormente, e o *software* real será submetido à engenharia com a devida análise de qualidade. Portanto, o cliente precisa entender que o protótipo é um executável (PRESSMAN, 2006).

Figura 10 – Modelo de Prototipagem.

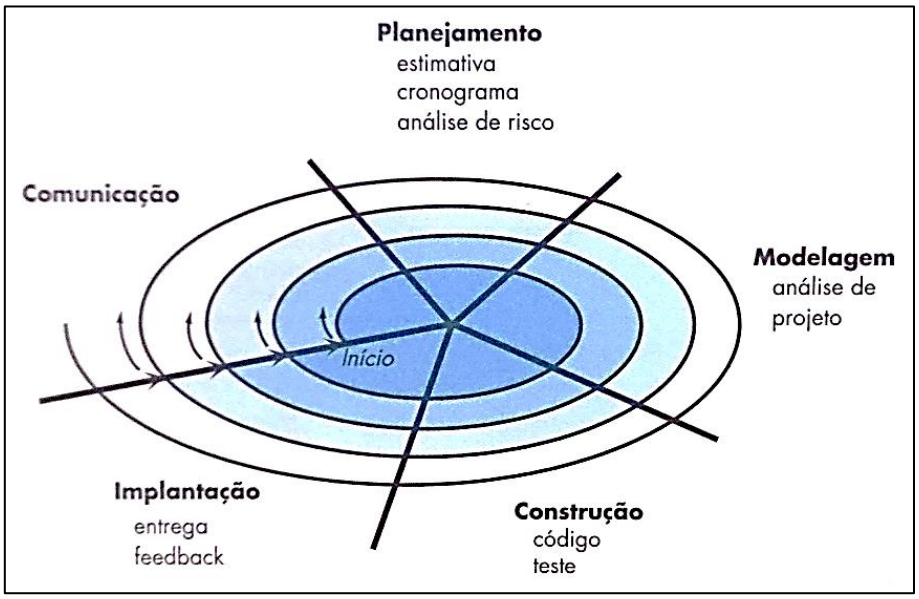


Fonte: PRESSMAN, 2006, p. 43.

O modelo em espiral combina a natureza iterativa da prototipagem com os aspectos controlados do modelo em cascata. Assim, ele segue uma abordagem cíclica que aumenta o grau de definição e de implementação de um sistema a cada incremento, ao passo que diminui o risco (BOEHM, 2001 apud PRESSMAN, 2006). O modelo pode ser aplicado ao longo de todo ciclo de vida de uma aplicação, sendo desenvolvido em uma série de versões evolucionárias.

Em vez de representar o processo de *software* como sequências de atividades com algum retorno entre uma atividade e outra, o processo é representado por uma espiral, onde cada volta da espiral representa uma fase do processo de *software*. Cada volta da espiral é dividida em cinco setores, conforme a Figura 11 (PRESSMAN, 2006).

Figura 11– Modelo em Espiral.

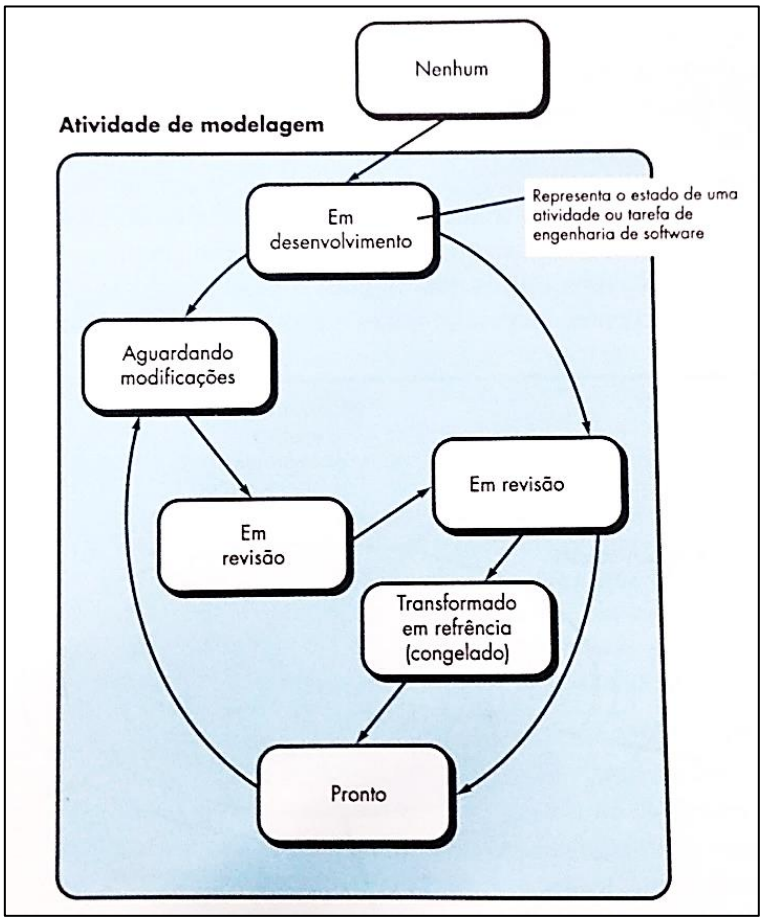


Fonte: PRESSMAN, 2006, p. 45.

O modelo de desenvolvimento concorrente estabelece que as atividades ocorram em paralelo, porém em diferentes estados. O modelo define uma série de eventos que vão disparar transições de estado para estado, para cada uma das atividades. Isto é, as atividades de comunicação, planejamento, modelagem, construção e implantação existem concomitantemente e podem estar situadas em qualquer estado da estrutura, conforme apresentada pela Figura 12 (PRESSMAN, 2006).

A intenção dos modelos evolucionários é desenvolver *softwares* de alta qualidade de maneira iterativa ou incremental. No entanto, os processos evolucionários enfatizam flexibilidade, extensibilidade e velocidade de desenvolvimento. O desafio para as equipes de *software* e seus gerentes é estabelecer um equilíbrio adequado entre esses parâmetros críticos de projeto e de produto e a satisfação do cliente, pois, em princípio, esses parâmetros são antagônicos ao alto padrão de qualidade (YOURDON, 1995; BACH, 1997 apud PRESSMAN, 2006).

Figura 12 – Modelo de Desenvolvimento Concorrente.



Fonte: PRESSMAN, 2006, p. 46.

2.1.2.3 Modelos Especializados

O modelo de desenvolvimento baseado em componentes é composto por aplicações de componentes de *software* previamente preparados, ou seja, por componentes de *softwares* comerciais de prateleira (COTS – *Commercial-off-the-shelf*). Para isso, é preciso seguir os seguintes passos (PRESSMAN, 2006):

1. Pesquisa e avaliação de componentes disponíveis para o domínio em questão;
2. Considerações sobre a integração de componentes;
3. Projeto de arquitetura de *software*;
4. Integração dos componentes à arquitetura;
5. Testes para garantir a funcionalidade adequada.

Portanto, esse modelo propicia o reuso de *software*, e este fato contribui para que o modelo de desenvolvimento baseado em componentes tenha o prazo do ciclo de desenvolvimento reduzido em 70% e o custo do projeto reduzido em 84% (YOURDON, 1994 apud PRESSMAN, 2006).

O modelo de métodos formais permite ao engenheiro de *software* especificar, desenvolver e verificar um sistema aplicando uma rigorosa notação matemática. Assim, é possível

descobrir e corrigir erros como ambiguidade, incompletude e inconsistência mais facilmente. Uma variante dessa abordagem chamada ‘engenharia de *software* sala limpa’ é usada atualmente e promete *softwares* livres de defeitos (MILLS, 1987; DYER, 1992 apud PRESSMAN, 2006). Contudo, o modelo possui desvantagens como:

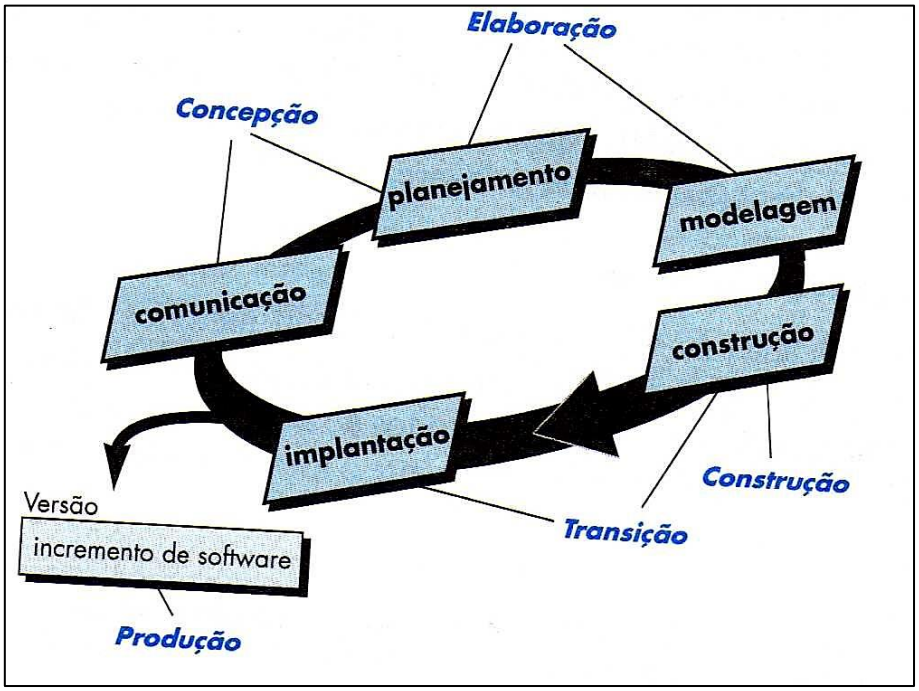
- Lentidão e necessidade de grande dispêndio de recursos;
- Exige treinamento extensivo aos desenvolvedores até que obtenham o preparo necessário;
- É difícil usar os modelos como um mecanismo de comunicação com a maioria dos clientes.

O modelo de desenvolvimento de *software* orientado a aspectos fornece mecanismos para definir, especificar, projetar e construir aspectos. Aspectos são preocupações do cliente que permeiam os diversos níveis do sistema, incluindo: propriedades de alto nível (ex.: segurança, tolerância a falha); funções (ex.: aplicação de regras de negócio); sistemáticas (ex.: sincronização e gestão de memória) (PRESSMAN, 2006).

2.1.2.4 Processo Unificado

Jacobson, Booch e Rumbaugh (1999 apud PRESSMAN, 2006) desenvolveram o Processo Unificado, um arcabouço para a engenharia de *software* que combina as melhores características e recursos dos modelos convencionais de modo a criar os princípios de desenvolvimento ágil de *software*. É importante ressaltar que este modelo é específico para projetos orientados a objetos (POO).

Figura 13 – Processo Unificado.



Fonte: JACOBSON; BOOCH; RUMBAUGH, 1999 apud PRESSMAN, 2006, p. 52.

A fase de concepção compreende as atividades de comunicação com o cliente, onde o analista irá fazer o primeiro contato com o cliente em busca das primeiras informações sobre o sistema a ser desenvolvido, com o objetivo de identificar os requisitos, definir o escopo do sistema; e de planejamento, onde o analista irá avaliar a viabilidade de implantação do sistema, dos recursos e dos riscos que serão gerenciados ao longo do projeto através da elaboração de um rascunho da arquitetura do sistema e pela definição do cronograma e do custo do projeto. A fase de elaboração inclui ainda as atividades de planejamento e também de modelagem do processo, pois nela ocorre a expansão e o refinamento dos casos de uso elaborados pela fase anterior, por meio da avaliação dos riscos técnicos e arquiteturais, incluindo cinco visões diferentes do *software*: o modelo de casos de uso, o modelo de análise, o modelo de projeto, o modelo de implementação e o modelo de implantação.

A fase de construção visa desenvolver os componentes de *software* que tornam os casos de uso operacionais para os usuários finais. Deste modo, a medida que os componentes são implementados, são realizados testes unitários, atividades de integração e testes de aceitação conforme as especificações dos casos de uso. A fase de transição envolve os últimos estágios do processo e a primeira etapa de implantação do *software* para o cliente, onde serão realizados testes com o objetivo de obter um feedback dos usuários em relação aos defeitos e/ou modificações necessárias, além de entregar informações de apoio, como manuais, guias de solução de problemas e procedimentos de instalação. Esta fase é responsável por, aproximadamente, 60% a 80% de todo o esforço despendido nos processos de desenvolvimento de *software*. E a fase de produção refere-se ao momento em que o incremento de *software* torna-se uma versão utilizável e de qualidade. (PRESSMAN, 2006).

2.1.3 GESTÃO DE PROJETOS DE SOFTWARE

Para Pressman (2006), o sucesso de um produto de *software* depende de seu gerente, que deve ser capaz de compreender que o trabalho de engenharia de *software* é um empreendimento intensamente humano, no qual ele precisa encorajar, desde o início do projeto a comunicação ampla com os interessados, além de atentar-se para o risco que o processo corre ao empregar, precipitadamente, alguns métodos e ferramentas técnicas. Portanto, é imperativo possuir um plano de projeto sólido. Por esses motivos, a gestão efetiva de projetos de *software* considera conceitos e princípios básicos (métricas, estimativa de projetos, cronogramação, riscos, qualidade e modificações) para gerenciar o pessoal, o produto, o processo e o projeto:

- Pessoal: deve ser organizado em equipes efetivas que estejam motivadas para fazer o trabalho de *software* com alta qualidade, e coordenado para alcançar a comunicação efetiva;

- Produto: depende dos requisitos informados pelo cliente, os quais serão decompostos e distribuídos para serem trabalhados pela equipe de *software*;
- Processo: deve ser adaptado às pessoas e ao problema de forma que uma estrutura geral de processo seja selecionada, para que um modelo de engenharia de *software* adequado possa ser aplicado, permitindo um conjunto de tarefas ser escolhido para a execução do serviço;
- Projeto: deve ser organizado de modo que leve a equipe de *software* ao sucesso.

Os princípios básicos mencionados acima são apresentados a seguir.

2.1.3.1 Métricas

As métricas de processo e de projeto são a base para a tomada de decisões gerenciais efetivas, pois são medidas quantitativas que permitem mensurar a eficácia do processo de *software*. Ao analisar esses dados numéricos, comparando-os às médias anteriores coletadas, é possível controlar o projeto, determinar se ocorreram melhorias de qualidade e produtividade e detectar possíveis áreas de problemas de modo que possam ser desenvolvidas soluções mais adequadas e mais céleres. Portanto, a métrica é uma ferramenta de gestão que fornece conhecimento ao gerente de projeto e a equipe de *software*, apoiando-os na condução do sucesso do projeto através do melhoramento dos processos de *software* (PRESSMAN, 2006).

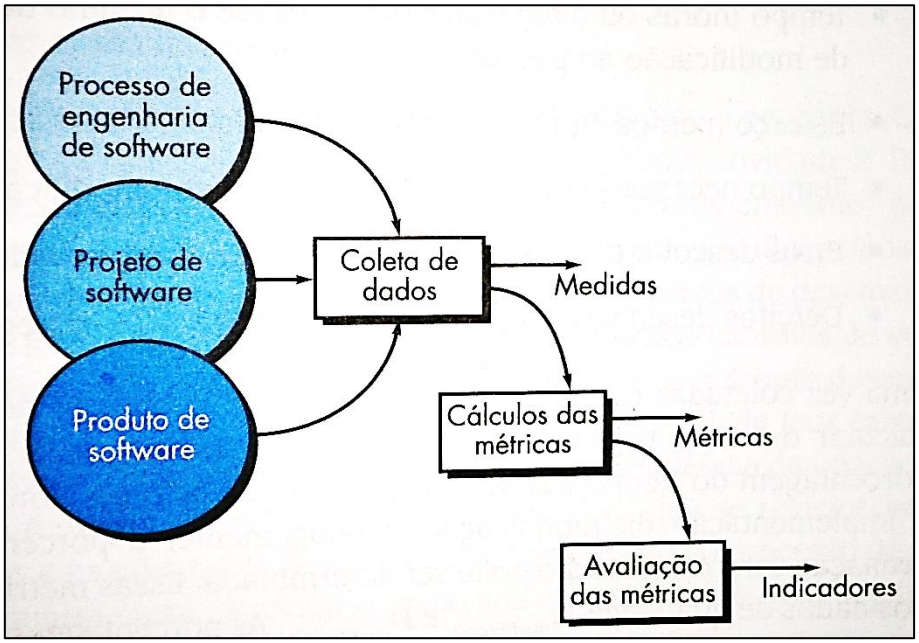
Para Fonseca e Rozenfeld (2012), um conjunto de métricas que quantifica tanto a eficiência, como a eficácia das ações é tido como um Sistema de Medição de Desempenho (SMD). E este sistema deve ser: preventivo, simples e claro, multifuncional, direcionado ao time, franco e visual (HAMERI; NIHTILA, 1998 apud FONSECA; ROZENFELD, 2012).

A medição de *software* é dividida em dois modos: medidas diretas (custo e esforço aplicados, linhas de código, velocidade de execução, tamanho de memória, defeitos detectados durante um período, etc.) e medidas indiretas (funcionalidade, qualidade, complexidade, eficiência, confiabilidade, manutenibilidade, etc.). Essas medidas ainda podem ser agrupadas em métricas de processo e de projeto (PRESSMAN, 2006):

- Métricas de processo: são coletadas no decorrer de todos os projetos e durante longos períodos; sua intenção é melhorar o processo de *software* em si, contribuindo para o desenvolvimento de métricas de projeto;
- Métricas de projeto: são coletadas durante a execução de cada projeto de modo que permite ao gerente de *software*: a avaliação do estado do projeto, o acompanhamento dos riscos potenciais, a descoberta de áreas problemas, o ajuste do fluxo de trabalho/tarefas e a avaliação da capacidade da equipe.

O processo de coleta, cálculo e avaliação de métricas de *software* exibido pela Figura 14, mostra que a partir da medição dos dados coletados dos processos, projetos e produtos de *software* e do cálculo e normalização das métricas, é possível estabelecer indicadores para a gestão de projetos de *software* (PRESSMAN, 2006).

Figura 14 – Processo de Coleta, Cálculo e Avaliação de Métricas de *Software*.



Fonte: PRESSMAN, 2006, p. 513.

2.1.3.2 Estimativa de Projetos

Antes do projeto começar, o gerente e a equipe de *software* precisam estimar o trabalho a ser feito, ou seja, quanto de dinheiro, esforço, recurso e tempo será necessário para o desenvolvimento do projeto de *software*. A estimativa sempre estará atrelada a um determinado grau de incerteza inevitável e deve considerar tanto o melhor, quanto o pior caso (PRESSMAN, 2006; BROOKS, 1975 apud PRESSMAN, 2006). Para isso, as estimativas de projetos de *software* devem abordar um conjunto de seis tarefas (PRESSMAN, 2006):

1. Estabelecer o escopo do projeto;
2. Determinar a viabilidade;
3. Analisar os riscos;
4. Definir os recursos necessários;
5. Estimar o custo e o esforço necessário;
6. Desenvolver um cronograma do projeto.

O escopo deve delimitar as funções e características a serem entregues aos usuários finais, os dados que entram e saem, o conteúdo, o desempenho, as restrições, as interfaces e a confiabilidade do sistema (PRESSMAN, 2006). Em seguida, é preciso analisar a viabilidade de construir um *software* que satisfaça a esse escopo. Para isso, avalia-se a viabilidade

tecnológica (o projeto é exequível tecnicamente?), a viabilidade financeira (o projeto será finalizado a um custo que a organização, o cliente ou o mercado possa pagar?), a viabilidade de tempo (o prazo para o lançamento do produto irá vencer a concorrência?), e a viabilidade de recursos (a organização tem os recursos necessários para o projeto?) (PUTNAM & MYERS, 1997 apud PRESSMAN, 2006).

O risco é um problema em potencial que afeta acontecimentos futuros, portanto, envolve a escolha e a incerteza da própria escolha. Por isso é importante tomar ações para modificá-lo: identificando-o, avaliando sua probabilidade de ocorrência, estimando seu impacto e estabelecendo um plano de contingência (PRESSMAN, 2006; CHARETTE, 1989 apud PRESSMAN, 2006).

Os recursos são caracterizados por quatro propriedades: descrição, disponibilidade, momento em que o recurso será necessário e o tempo (duração) no qual o recurso será aplicado; e são divididos em três categorias: humanos, de *software* reusáveis e de ambiente. Os recursos humanos são analisados quanto às habilidades, a quantidade e a posição na organização (gerente, engenheiro, técnico) do pessoal. Os recursos de *software* reusáveis se referem aos componentes de *software* do projeto, que podem ser: componentes de prateleira (foram adquiridos de terceiros e estão prontos e validados), componentes de experiência plena (foram desenvolvidos para projetos anteriores, porém, por serem similares ao projeto atual, podem ser usados sem muitas modificações), componentes de experiência parcial (também foram desenvolvidos para projetos anteriores, contudo, exigirão substancial modificação) ou componentes novos (precisam ser construídos especificamente para o projeto atual). E os recursos de ambiente apoiam o projeto de desenvolvimento de *software*, sendo formado pelo hardware, pelas ferramentas de *software* e pelos recursos de rede (PRESSMAN, 2006; BENNATAN, 1992 apud PRESSMAN, 2006).

O custo e o esforço necessário só devem ser estimados a partir de projetos anteriores quando o projeto atual for bastante semelhante ao precedente. Senão, é recomendável usar técnicas de decomposição e modelos empíricos. Na decomposição, o problema ou o processo é subdividido em partes menores, reduzindo o *software* em funções que possam ser estimadas. E podem ser calculados por estimativa de três pontos (otimista, mais provável e pessimista), número de linhas de códigos, quantidade de pontos de função ou uma combinação deles (PRESSMAN, 2006; PUTNAM & MYERS, 1997 apud PRESSMAN, 2006). Dados os valores estimados para o tamanho do *software* através do número de linhas de código (LOC) ou quantidade de pontos de função (FP), os modelos empíricos estimam o esforço a partir da regressão de dados coletados em projetos de *software* anteriores e possuem a seguinte estrutura (MATSON, 1994 apud PRESSMAN, 2006):

$$E = A + B\ x\ (ev)^C$$

A, *B* e *C* são constantes derivadas empiricamente, *E* é o esforço em pessoas-mês e *ev* é a variável de estimativa (LOC ou PF) (MATSON, 1994 apud PRESSMAN, 2006).

Quadro 1 – Modelos Empíricos de Estimativa de Projetos.

AUTOR	MODELO
Walston-Felix	$E = 5{,}2 \times (KLOC)^{0{,}91}$
Bailey-Basili	$E = 5{,}5 + 0{,}73 \times (KLOC)^{1{,}16}$
Boehm	$E = 3{,}2x\ (KLOC)^{1{,}05}$
Doty	$E = 5{,}288x\ (KLOC)^{1{,}047}$
Albrecht-Gaffney	E = -91,4 + 0,355 x (FP)
Kemerer	E = -37 + 0,96 x (FP)

Fonte: PRESSMAN, 2006, p. 534.

O cronograma do projeto só pode ser desenvolvido após a seleção do modelo de processo adequado, a identificação das tarefas de engenharia de *software* a serem realizadas, a estimativa da quantidade de trabalho e do número de pessoas necessário, os prazos e os riscos terem sido considerados. Pois, somente assim será possível construir um cronograma realístico para o desenvolvimento de um plano de projeto efetivo (PRESSMAN, 2006).

2.1.3.3 Cronogramação

Definir um cronograma é criar uma rede de tarefas de engenharia de *software* que irá garantir que o serviço seja entregue no prazo. Para isso, o gerente deve: definir todas as tarefas do projeto, construir uma rede que mostre as correlações entre as tarefas, identificar o caminho crítico dessa rede e acompanhar o progresso das tarefas, certificando-se de que atrasos sejam reconhecidos prontamente. Além de ser uma atividade que distribui o esforço estimado pela duração planejada do projeto (PRESSMAN, 2006).

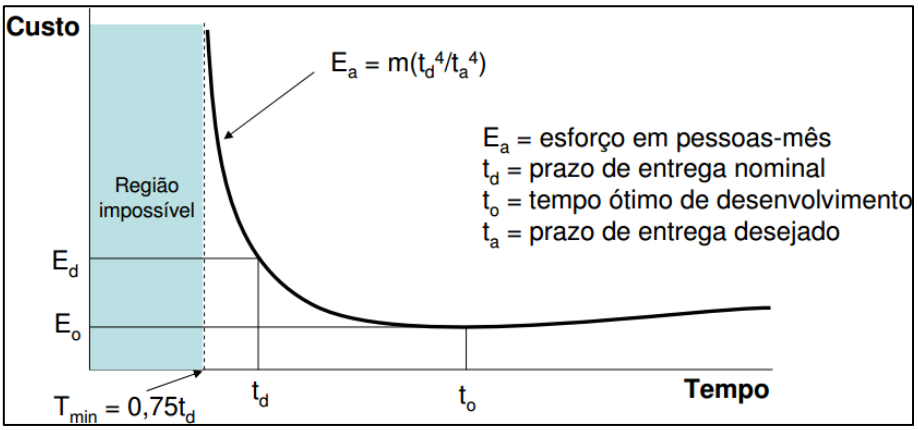
Pressman (2006) diz que a cronogramação é orientada por sete princípios básicos:

- Compartimentalização: decomposição do projeto em tarefas e atividades gerenciáveis;
- Interdependência: correlação entre as atividades, ações ou tarefas em relação ao fluxo do projeto de forma que existirão atividades que ocorrerão em sequência, enquanto outras ocorrerão em paralelo;
- Atribuição de tempo: definição da unidade de trabalho e das datas de início e término de cada tarefa;
- Validação do esforço: garantia do gerente de projeto de que o esforço exigido por determinado trabalho não seja maior do que o número disponível de pessoas alocadas para aquela tarefa;

- Responsabilidades definidas: designação de um responsável da equipe para cada tarefa;
- Resultados definidos: estabelecimento de um resultado de trabalho para cada tarefa;
- Marcos de referência definidos: demarcação de um evento de referência para o cumprimento de um ou mais produtos do trabalho.

O cronograma de um projeto é empiricamente elástico, ou seja, é possível comprimir, até certo ponto, a data de entrega de um determinado projeto através do acréscimo de recursos adicionais. Igualmente, é exequível estender a data de entrega por meio da redução do número de recursos. Assim sendo, a Figura 15 apresenta o relacionamento entre o esforço aplicável e o prazo de entrega de um projeto de *software*.

Figura 15 – Curva PNR (Putnam-Norden-Rayleigh).



Fonte: NORDEN, 1970 apud PRESSMAN, 2006; PUTNAM, 1978 apud PRESSMAN, 2006, p. 548.

A rede de tarefas é uma representação gráfica do fluxo de atividades de um projeto, na qual as tarefas e subtarefas têm interdependências baseadas na sua sequência. Assim, as tarefas devem ser coordenadas para que sejam completadas quando tarefas posteriores precisarem do seu produto de trabalho, por isso a importância do caminho crítico (cadeia de tarefas que determina a duração do projeto) (PRESSMAN, 2006).

O acompanhamento do projeto pode ser feito de diversas maneiras a depender da preferência do gerente de projetos, seja através de reuniões periódicas de avaliação do estado do projeto, onde cada membro da equipe relata o progresso e os problemas, ou pelos marcos de referência atingidos na data prevista, comparando a data de início real com a data de início prevista para cada tarefa, ou por meio da análise do valor agregado, avaliando o progresso quantitativamente. No entanto, quando ocorrem problemas, o gerente de projeto deve exercer controle para resolvê-los o mais rápido possível. Para isso é preciso: diagnosticar o problema, alocar recursos adicionais, redistribuir o pessoal e redefinir o cronograma (PRESSMAN, 2006).

Ludwig, Anzanello e Vidor (2013) propuseram um estudo para reduzir atrasos nas tarefas de uma empresa de desenvolvimento do software a partir do uso de sequenciamento de tarefas. Para isso, duas heurísticas distintas (H1 e H2) foram apresentadas e aplicadas em cenários que replicavam características semelhantes ao da empresa em estudo. Na sequência, as heurísticas foram aplicadas no cenário real da empresa, e os resultados de atraso total foram comparados com o sequenciamento manual realizado pelo gerente responsável.

Quadro 2 – Heurísticas Utilizadas no Sequenciamento das Atividades de Desenvolvimento de *Software*.

Heurística	Passo 1: Ordenamento inicial das tarefas	Passo 2: Alocação das tarefas aos times de desenvolvimento	Passo 3: Sequenciamento das tarefas em cada time
H1	Tempo de execução mais curto	Tempo de processamento acumulado	Minimização do atraso total através da heurística baseada em programação dinâmica de Heugler e Vasko (1997)
H2	Menor folga		

Fonte: LUDWIG; ANZANELLO; VIDOR, 2013, p. 489.

A heurística H2 apresentou melhores resultados tanto ao minimizar o atraso total nos dados simulados, como em relação ao sequenciamento das tarefas reais. E quando comparadas ao sequenciamento manual, H2 obteve uma redução de 21% no atraso total, enquanto que H1 manteve o mesmo atraso (LUDWIG; ANZANELLO; VIDOR, 2013). Portanto, a cronogramação deve atentar-se mais a redução das folgas entre as atividades, do que a redução do tempo de execução das atividades.

2.1.3.4 Gestão de Riscos

A gestão de risco envolve a identificação dos riscos, a estimativa da probabilidade de ocorrência, a avaliação do impacto e o estabelecimento de um plano de contingência. Desse modo, as estratégias para a gestão de risco podem ser proativas ou reativas, dependendo da natureza do risco: conhecido, previsível ou imprevisível (PRESSMAN, 2006).

Assim sendo, existem três tipos de risco de *software*:

- Riscos de projeto: ameaçam o plano do projeto de tal forma que, provavelmente, irão atrasar o cronograma e elevar os custos do projeto. Por isso é importante identificar os impactos no orçamento, no cronograma, no pessoal (quantidade e organização), nos recursos, nos interessados e nos requisitos;
- Riscos técnicos: ameaçam a qualidade do *software*, tornando a implementação difícil ou até impossível. Sendo assim, é preciso identificar os problemas em potencial no projeto, na implementação, na interface, na verificação e na manutenção;
- Riscos de negócio: ameaçam a viabilidade do *software*, comprometendo o projeto ou o produto. Devendo então, atentar-se ao risco de mercado, risco estratégico, risco de vendas, risco gerencial e risco orçamentário.

A etapa de identificação dos riscos é uma tentativa sistemática de especificar os riscos conhecidos e previsíveis, considerando tanto os riscos genéricos, como os riscos específicos do produto. Para isso, é indicado utilizar *checklists* dos itens de risco (tamanho do produto, impacto no negócio, características do cliente, definição do processo, ambiente de desenvolvimento, tecnologia para a construção, tamanho e experiência da equipe) ou matrizes de avaliação de impacto (PRESSMAN, 2006). O impacto de um fator de risco pode ser categorizado como: catastrófico, crítico, marginal ou negligível de acordo com a consequência do risco em relação ao desempenho, suporte, custo e cronograma conforme apresentado pela Figura 16 (BOEHM, 1989 apud PRESSMAN, 2006).

Figura 16 – Matriz Avaliação de Impacto.

Componentes / Categoria		Desempenho	Apoio	Custo	Cronograma
Catastrófica	1	Falha em satisfazer o requisito resultaria na falha da missão		A falha resulta em aumento de custo e atraso de cronograma com valores previsto que excedem $500 mil	
	2	Da degradação significativa a não-alcance do desempenho técnico	Apoio de software não-responsivo ou software inapoiável	Falta significativa de recursos financeiros, provável estouro de orçamento	Data de entrega inexeqüível
Crítica	1	Falha em satisfazer o requisito degradaria o desempenho do sistema até um ponto em que o sucesso da missão é questionável		A falha resulta em atrasos operacionais e/ou aumento de custos com valor previsto de $100 a $500 mil	
	2	Alguma redução no desempenho técnico	Pequenos atrasos nas modificações do software	Alguma falta de recursos financeiros, possível estouro de orçamento	Possível ultrapassagem da data de entrega
Marginal	1	Falha em satisfazer o requisito resultaria na degradação da missão secundária		Custos, impactos e/ou atrasos de cronograma recuperáveis com valor previsto de $1 a $100 mil	
	2	De mínima a pequena redução do desempenho técnico	Apoio de software responsivo	Recursos financeiros suficientes	Cronograma realístico exeqüível
Negligível	1	Falha em satisfazer o requisito criaria inconveniência ou impacto não operacional		Erro resulta em pequeno impacto no custo e/ou cronograma com valor previsto de menos que $1 mil	
	2	Sem redução no desempenho técnico	Software facilmente apoiável	Possível sobra de orçamento	Data de entrega antecipável

Fonte: BOEHM, 1989 apud PRESSMAN, 2006, p. 567.

A segunda etapa se refere à estimativa da probabilidade de ocorrência e envolve quatro atividades: estabelecer uma escala que reflita a probabilidade percebida do risco, esboçar as consequências do risco, estimar o impacto do risco no projeto e no produto, e calcular a precisão da previsão do risco. Feito isso, deve-se organizar essas informações em uma tabela de risco com cinco colunas de forma que ela seja classificada em ordem decrescente do produto entre probabilidade e impacto. A primeira coluna identifica o risco, a segunda indica o tipo de risco, a terceira indica a probabilidade do risco, a quarta indica o impacto de cada risco e a quinta coluna destina-se ao plano de mitigação. É importante que o gerente

estabeleça uma linha de corte para a tabela de risco, pois é impossível monitorar e controlar todos os riscos, portanto, somente os riscos acima da linha receberão atenção subsequente (PRESSMAN, 2006).

Segundo Pressman (2006), a próxima etapa é de avaliação do impacto do risco, a qual depende de três fatores: sua natureza (indica os tipos de problemas que podem surgir se ele ocorrer), seu escopo (combina a severidade com o quanto o projeto será afetado) e sua época (quando e por quanto tempo seu impacto será sentido).

Isto posto, pode ser calculada uma métrica de exposição ao risco (RE) através do produto da probabilidade de ocorrência do risco (P) e do custo para o projeto caso o risco ocorra (C). Essa métrica determina o custo esperado do risco (HALL, 1998 apud PRESSMAN, 2006).

$$RE = P\ x\ C$$

A última etapa se refere ao estabelecimento de um plano de contingência, para isso deve-se buscar atenuar, monitorar e gerenciar o risco. Essa estratégia pode ser tratada por um plano RMMM (*Risk Mitigation, Monitoring and Management*), o qual documenta todo o trabalho realizado como parte da análise de risco, porém isto implica num custo adicional para o projeto, portanto, o gerente deve avaliar quando os benefícios obtidos pelos RMMM são superiores aos seus custos. Logo, há riscos que não compensam serem levados ao RMMM, cabendo apenas monitorar o risco e não o atenuar (PRESSMAN, 2006).

2.1.3.5 Gestão da Qualidade

Segundo Pressman (2006), a gestão da qualidade de *software* depende de um conjunto de atividades que garanta a todos os produtos de trabalho da engenharia de *software*, alta qualidade. Para isso, é preciso realizar atividades de controle em todo o projeto por meio de métricas que permitam o aperfeiçoamento do processo de *software*, e como consequência, a qualidade do produto final. Assim sendo, a gestão da qualidade de *software* se resume em cinco características:

1. Processo de garantia de qualidade de *software*;
2. Tarefas especificas de garantia e controle da qualidade;
3. Prática de engenharia de *software* efetiva;
4. Controle de todos os produtos de trabalho de *software* e de suas modificações;
5. Procedimento de garantia da satisfação de normas de desenvolvimento de *software*, mecanismos de medição e relatório.

Glass (1998 apud PRESSMAN, 2006) ressalta que a qualidade é importante, porém a satisfação do cliente é mais importante do que isso e depende de o produto estar adequado à sua necessidade com a máxima qualidade, sendo entregue dentro do orçamento e do

cronograma. Por esse motivo, o controle, a garantia e o custo da qualidade são tão importantes, de modo que devam buscar:

- Minimizar a diferença entre os recursos previstos e os recursos usados (pessoal, equipamento e tempo);
- Minimizar a variância no número de erros de uma versão para outra;
- Minimizar as diferenças na velocidade e na precisão de respostas aos problemas de clientes.

O controle da qualidade abrange uma série de inspeções, revisões e testes ao longo do projeto de *software* para avalizar que cada produto de trabalho satisfaça aos requisitos estabelecidos para ele, pois todos os produtos de trabalho devem ter especificações mensuráveis para permitir a comparação do resultado de cada processo. A garantia da qualidade consiste num conjunto de auditorias e relatórios que avaliam a efetividade das atividades de controle da qualidade, auxiliando a gerência na identificação de problemas e posterior aplicação adequada dos recursos para resolução dessas questões sobre a qualidade do produto. O custo da qualidade compreende todo e qualquer custo relacionado à busca da qualidade ou da execução das atividades relacionadas à qualidade, isso envolve: custos de prevenção, custos de avaliação, custos de falha interna e externa (PRESSMAN, 2006).

A garantia da qualidade de *software* ou SQA (*Software Quality Assurance*) visa atingir a conformidade dos requisitos funcionais e de desempenho (base pela qual a qualidade é medida), as normas de desenvolvimento explicitamente documentadas (conjunto de critérios que guia o modo pelo qual o *software* é submetido à engenharia) e outras características implícitas (facilidade de uso e boa manutenibilidade). Isto posto, um plano de SQA deve fornecer um guia para a instituição garantir a qualidade de *software* e pode ser confirmado estatisticamente pelo princípio de Pareto (diz que 80% dos defeitos podem ser explicados por até 20% de todas as causas possíveis) (PRESSMAN, 2006). O princípio de Pareto estipula que: (1) as informações a respeito dos defeitos sejam coletas e categorizadas; (2) seja rastreada a causa de cada defeito; (3) seja isolado os 20% “poucos vitais”; e (4) os problemas que causaram os defeitos dos 20% “poucos vitais” sejam corrigidos.

Semelhantemente, o SQA também pode ser estatisticamente comprovado através da metodologia Seis Sigma, a qual define três principais pontos (PRESSMAN, 2006):

- Definição dos objetivos do projeto, dos requisitos do cliente e dos artefatos de entrega através da comunicação com o cliente;
- Medição do processo e das saídas dele para determinar o atual desempenho de qualidade;
- Análise das métricas de defeito e determinação das poucas causas vitais.

O método DMAIC (*Define, Measure, Analyse, Improve and Control*) é o mesmo que o Seis Sigma quando um processo de *software* já está em execução e necessita de aprimoramento (PRESSMAN, 2006). Para isso, o Seis Sigma adiciona mais dois pontos ao modelo:

- Aperfeiçoamento do processo através da eliminação das causas básicas dos defeitos;
- Controle do processo que garanta ao trabalho futuro, a não reintrodução das causas desses defeitos.

Caso o processo de *software* não necessite de aprimoramento, o método DMADV (*Define, Measure, Analyse, Design and Verify*) é mais indicado (PRESSMAN, 2006), pois envolve:

- Projeção do processo para evitar as causas básicas de defeitos e para satisfazer os requisitos do cliente;
- Verificação da capacidade do modelo do processo para evitar as causas básicas de defeitos e satisfazer aos requisitos do cliente.

2.1.3.6 Gestão das Modificações

As modificações são inevitáveis e podem ocorrer em qualquer etapa durante um projeto de *software*. Portanto, o seu gerenciamento é indispensável e deve ocorrer de maneira controlada e efetiva, pois quando não controladas, as modificações costumam onerar significativamente os projetos. Assim, a gestão de modificações ou SCM (*Software Configuration Management*) deve: identificar, controlar e garantir a adequada implementação e relatar aos interessados as modificações. As modificações advêm basicamente de quatro fontes (PRESSMAN, 2006):

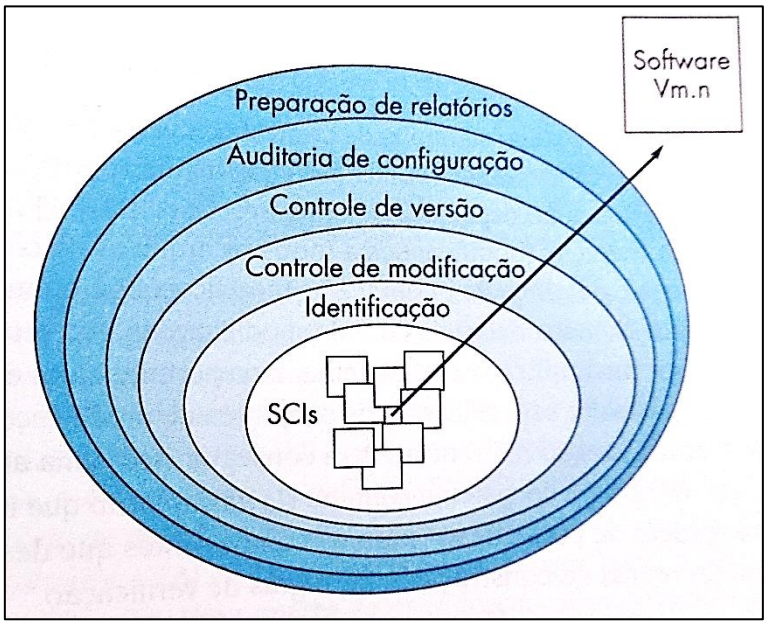
- Novas condições de negócio ou de mercado que geram novos requisitos ou novas regras de negócio;
- Novas necessidades do cliente que exigem novos dados, funcionalidades ou serviços;
- Reorganização do negócio que causa alterações nas prioridades do projeto ou na estrutura da equipe;
- Restrição de orçamento ou cronograma que causam redefinição do sistema ou do produto.

Dart (2001) define quatro importantes elementos presentes durante o desenvolvimento de um sistema de SCM: elementos humanos, de componente, de processo e de construção. Esses elementos influenciam os itens de configuração de *software* (*Software Configuration Index* – SCIs), que são formados por programas de computador (código-fonte e executável), produtos de trabalho (técnicos de *software* e usuários) e dados (do programa ou externos a ele). Quando algum desses itens é formalmente revisto, aprovado e armazenado, ele se

torna um referencial de forma que daí em diante ele irá servir como base para o desenvolvimento futuro (IEEE, 1990 apud PRESSMAN, 2006). Os SCIs são mantidos em um banco dedados (repositório), o qual permite gerir as modificações de modo efetivo, pois garante a integridade de dados, o compartilhamento de informações, a integração de ferramentas, a integração de dados, a imposição de metodologia e a padronização de documentação (FORTE, 1989 apud PRESSMAN, 2006).

O processo de gestão de configuração de *software* (SCM) possui quatro objetivos principais: (1) identificar todos os itens que definem coletivamente a configuração de *software*; (2) gerir modificações em um ou mais desses itens; (3) facilitar a construção de diferentes versões de uma aplicação e (4) garantir que a qualidade do *software* seja mantida à medida que a configuração evolui ao longo do tempo. E, é visto como um modelo de cinco camadas concêntricas: identificação, controle de versão, controle de modificação, auditoria de configuração e preparação de relatórios; onde os SCIs fluem de dentro para fora dessas camadas durante sua vida útil, de modo que ao final, torne-se parte da configuração de *software* de uma ou mais versões de uma aplicação ou sistema (PRESSMAN, 2006).

Figura 17 – Camadas do Processo de SCM.



Fonte: PRESSMAN, 2006, p. 607.

Na camada de identificação, cada item da configuração deve receber um nome e ser gerenciado usando uma abordagem orientada a objetos. Cada objeto tem um conjunto de atributos que o identificam unicamente (nome, descrição, lista de recursos e um ponteiro para o produto de trabalho). O controle de modificação é vital, pois visa impedir que uma pequena perturbação no código crie uma grande falha para o produto. Porém, esse processo é burocrático, porque todas as modificações devem ser acompanhadas e revisadas antes de serem aprovadas. Já a camada de controle de versão combina

procedimentos e ferramentas para gerir diferentes versões de objetos de configuração, e deve possuir quatro capacidades principais: (1) armazenar todos os objetos de configuração relevantes no repositório; (2) gerir a versão que guarda todas as versões de um objeto de configuração; (3) propiciar ao engenheiro de *software* coletar todos os objetos de configuração relevantes e gerar uma versão específica do *software* e (4) acompanhar o estado de todos os tópicos importantes relacionados a cada objeto.

A camada da auditoria de configuração propõe-se a verificar se a modificação foi adequadamente implementada por meio de revisões técnicas formais. Em outras palavras, examina se os procedimentos SCM foram seguidos. Por fim, a camada de preparação de relatórios se destina a manter a gerência e os profissionais informados de modificações importantes através de relatórios gerados regularmente, os quais devem descrever: o ocorrido, o autor, o momento ocorrido e o que mais poderá ser afetado (PRESSMAN, 2006).

2.1.4 DESENVOLVIMENTO ÁGIL DE SOFTWARE

As definições modernas de desenvolvimento de software ágil evoluíram a partir da metade dos anos 1990 como parte de uma reação contra os métodos pesados (tradicionais), caracterizados pela densa regulamentação, burocracia, lentidão e pelo micro gerenciamento. Nesse tempo, os métodos ágeis ainda eram conhecidos como métodos leves. A mudança ocorreu somente em 2001, na cidade de Snowbird, Utah, Estados Unidos, através da reunião de 17 pessoas que se propuseram a discutir sobre o assunto, entre eles haviam, além de simpatizantes ao tema, representantes de diversas metodologias como: *Extreme Programming* (XP), SCRUM, *Dynamic Systems Development Method* (DSDM), *Adaptive Software Development* (ASD), *Crystal*, *Feature-Driven Development* (FDD), *Pragmatic Programming*. Com fruto desse encontro, surgiu a publicação do documento "Manifesto Ágil", o qual consiste em 4 valores fundamentais e 12 princípios de desenvolvimento para esta nova metodologia (CUNNINGHAM, 2001).

Portanto, a gestão ágil diz respeito à ciência do gerenciamento aplicada à Engenharia de Software, por meio de uma abordagem mínima e leve para o gerenciamento, a qual utiliza um estilo de baixa intervenção e alta delegação de responsabilidade, fornecendo *empowerment* (ocorre quando um indivíduo aceita a responsabilidade pela entrega de uma saída com valor agregado, e para isso, precisa determinar o que é necessário e trabalhar com outros para conceber a melhor maneira para entregar tal saída com o mínimo de esforço) em todos os níveis. Um gerente ágil deve ser fortalecido, seus líderes de equipes devem ser fortalecidos, mas, sobretudo, os desenvolvedores e testadores que fazem o trabalho real devem ser fortalecidos (CUNNINGHAM, 2001).

Com o Manifesto é evidente perceber que as abordagens tradicionais são voltadas à geração de documentos e à sequência rígida dos processos. Em ambientes estáticos, isso

pode funcionar muito bem, porém, em cenários onde constantemente os requisitos são alterados a fim de possibilitar a adaptação dos produtos e/ou serviços às necessidades dos clientes, as metodologias ágeis são recomendadas. Assim, os métodos ágeis têm desempenhado um papel fundamental para o desenvolvimento de software moderno ao priorizar o valor que o projeto agrega e as interações entre as pessoas, face ao cumprimento de prazos, custos ou atendimento ao escopo inicialmente definido. De forma que os profissionais têm se tornado mais completos, os produtos têm sido desenvolvidos com mais qualidade e o clientes têm ficado mais satisfeitos (PRIKLADNICKI; WILLI; MILANI, 2014). De maneira geral, pode-se comparar as metodologias tradicional e ágeis da seguinte maneira (Figura 18).

Figura 18 – Metodologias de Desenvolvimento de Software.

	TRADICIONAL	METODOLOGIAS ÁGEIS
Pressupostos fundamentais	Sistemas totalmente especificáveis, previsíveis; desenvolvidos a partir de um planejamento extensivo e meticuloso	Software adaptativo e de alta qualidade; pode ser desenvolvido por equipes pequenas utilizando os princípios da melhoria contínua do projeto e testes orientados a rápida resposta a mudanças
Controle	Orientado a processos	Orientado a pessoas
Estilo de gerenciamento	Comandar e controlar	Liderar e colaborar
Gestão do conhecimento	Explícito	Tácito
Atribuição de papéis	Individual – favorece a especialização	Times auto-organizáveis – favorece a troca de papéis
Comunicação	Formal	Informal
Ciclo do projeto	Guiado por tarefas ou atividades	Guiado por funcionalidades do produto
Modelo de desenvolvimento	Modelo de ciclo de vida (Cascata, Espiral, ou alguma variação)	Modelo iterativo e incremental de entregas
Forma/estrutura organizacional desejada	Mecânica (burocrática com muita formalização)	Orgânica (flexível e com incentivos a participação e cooperação social)

Fonte: PRIKLADNICKI; WILLI; MILANI, 2014.

Portanto, a Figura 18 corrobora para mostrar que o ambiente de desenvolvimento ágil de *softwares* visa a desburocratização, a agilidade e a iteração entre os diversos envolvidos, de modo que possibilita responder rapidamente a mudanças. Sendo assim, a seguir é apresentado três dos principais modelos de desenvolvimento ágil de projetos de *software*: SCRUM, XP e FDD.

2.1.4.1 SCRUM

O SCRUM é uma metodologia ágil que auxilia no gerenciamento de projetos complexos e no desenvolvimento do produto atuando de forma iterativa, objetiva e incremental, trazendo

uma nova dimensão na capacidade de resposta e adaptabilidade da gestão dos processos (SCHWABER, 2004 apud PRIKLADNICKI; WILLI; MILANI, 2014). Audy (2015) sugere que o SCRUM seja utilizado no gerenciamento de projetos com equipes ágeis de pequeno porte, multidisciplinares, auto organizados e com foco na melhoria contínua.

Para Schwaber e Sutherland (2013), o SCRUM trata-se de um *framework* que integra papéis, eventos, artefatos e regras, as quais não define minuciosamente as atividades que devem ser feitas, mas elenca valores, princípios e práticas que são consideradas adequadas para a gestão da equipe, são elas: transparência, inspeção e adaptação. Isso reforça que a equipe deve se comunicar com clareza e sinceridade entre si a fim de que haja confiança entre ela, além de padrões definidos de compartilhamento de informações; deve, frequentemente, inspecionar os artefatos e se o progresso caminha em direção aos objetivos. Porém, esta inspeção não deve ser tão frequente que atrapalhe a própria execução das tarefas; e a equipe deve possuir uma capacidade adaptativa para absolver as mudanças necessárias para a melhoria.

O SCRUM se aplica bem a natureza de alta variação dos requisitos e de alto grau de imprevisibilidade dos projetos de desenvolvimento de *software*, conseguindo maximizar a entrega de *software* de modo eficaz, adaptando-se à realidade das mudanças. Para isso, desenvolve-se, primeiramente, as funcionalidades de maior valor, ao passo que se cogita sobre a necessidade ou não de desenvolvimento das menos prioritárias (PRIKLADNICKI; WILLI; MILANI, 2014).

Carvalho e Mello (2009) identificaram nove benefícios provenientes da utilização do SCRUM a partir de um levantamento de citações na literatura. Os três mais representativos (56% das aparições) foram:

- Melhoria na comunicação e aumento da colaboração entre envolvidos;
- Melhoria da qualidade do produto produzido;
- Aumento de produtividade da equipe.

Para que esses benefícios sejam alcançados, é necessário avaliar os requisitos que definem os produtos que serão desenvolvidos a fim de agregar valor ao cliente. No SCRUM, esses requisitos são apresentados no *Product Backlog*, documento que descreve o trabalho a ser realizado para a elaboração do produto. O *Product Backlog* é dinâmico, pois evolui tanto quanto o produto e o ambiente no qual ele será utilizado evoluem, mudando constantemente para identificar o que o produto necessita para ser mais apropriado, competitivo e útil. Há ainda, o *Sprint Backlog*, conjunto de itens do *Product Backlog* selecionados para serem realizados durante uma *Sprint* (espaço de tempo previamente definido no qual a equipe produz o incremento no produto), bem como o trabalho necessário para entregar esse

incremento “Pronto”, ou seja, em condições de utilização pelo cliente (SCHWABER; SUTHERLAND, 2013).

Schwaber e Sutherland (2013) definem o *Scrum Team*, como o grupo composto pelo *Product Owner*, o Time de Desenvolvimento e o *Scrum Master*. O responsável por gerenciar o *Product Backlog* é o *Product Owner*, o qual deve expressar e ordenar os itens que irão compor o *Sprint Backlog*, além de garantir que estes itens estejam claros para o Time de Desenvolvimento (profissionais que realizam o trabalho de entregar uma versão utilizável que potencialmente incrementa o produto ao final de cada *Sprint*). Este time, pautado no autogerenciamento, tem autonomia de escolher a melhor forma de realizar suas atividades, com o auxílio do *Scrum Master*, que possui o papel de garantir o entendimento (teoria, práticas, regras e interações) do SCRUM por parte de todos os envolvidos, além da sua correta aplicação.

O último elemento que compõe o SCRUM são os eventos, estes possuem duração fixa e são realizados em intervalos regulares, os quais permitem uma oportunidade para inspeção e adaptação. Os eventos do SCRUM são: *Sprint Planning* (Reunião de Planejamento da *Sprint*), *Daily Scrum* (Reunião Diária), *Sprint Review* (Revisão da *Sprint*), e *Sprint Retrospective* (Retrospectiva da *Sprint*) (PRIKLADNICKI; WILLI; MILANI, 2014).

- *Sprint Planning*: encontro inicial onde o *Scrum Team* define as entregas do incremento (funcionalidades que serão desenvolvidas) desta *Sprint* que se iniciará. Essas reuniões não podem passar de 8 horas de duração para uma *Sprint* de um mês. Para *Sprints* menores, o tempo é reduzido proporcionalmente;
- *Daily Scrum*: consiste em reuniões diárias do Time de Desenvolvimento com duração de até 15 minutos, onde cada membro responde a três perguntas: o que fiz desde a última *Daily Scrum*? O que pretendo fazer até a próxima *Daily Scrum*? E, existe algo me impedindo de concluir alguma tarefa?;
- *Sprint Review*: reunião realizada ao final de cada *Sprint* com o *Scrum Team* e qualquer outro interessado que queira participar. Além de apresentar as funcionalidades desenvolvidas durante a *Sprint*, esta reunião também prevê a inspeção e possíveis adaptações necessárias ao produto para as próximas *Sprints*;
- *Sprint Retrospective*: esta reunião ocorre imediatamente após a *Sprint Review* e visa aprimorar o processo, identificando melhorias que possam ser implementadas em relação às interações, práticas e ferramentas utilizadas.

2.1.4.2 Extreme Programming

Segundo MELO (2010) a *Extreme Programming* (XP), Programação Extrema em português, propõe um conjunto de valores, princípios e práticas que visam garantir o sucesso no desenvolvimento de *software* entregando produtos de alta qualidade com alta produtividade.

Contudo, a metodologia deixa a desejar por conta dos requisitos vagos e do alto grau de incerteza. Para isso, a metodologia XP se baseia em cinco valores fundamentais: comunicação, simplicidade, feedback, coragem e respeito; e 14 princípios (humanidade, economia, melhoria, benefício mútuo, semelhança, diversidade, passos pequenos, reflexão, fluxo, oportunidade, redundância, falha, qualidade e aceitação da responsabilidade) que funcionam como canais mentais para transformar os valores, que são abstratos em práticas de fácil implementação no dia a dia do desenvolvimento (PRIKLADNICKI; WILLI; MILANI, 2014). O sucesso da XP decorre da sinergia de seus valores e princípios, pois os pontos fracos de cada um deles são superados pelos pontos fortes dos outros.

Na Programação Extrema, a participação do cliente é imprescindível, e por isso, deve estar disposto a colaborar com a construção do *software*. Em geral, os desenvolvedores tomam decisões sobre as características do *software* a cada 15 minutos, e se não souberem a resposta, ou solicitam a informação ao cliente e a aguardam, ou assumem uma resposta como verdade e continuam. No primeiro caso, o desperdício e a queda da produtividade são inevitáveis, e no segundo, a chance de retrabalho é sempre latente. Portanto, a comunicação com o cliente deve ser rápida e eficiente através de *feedbacks* constantes, no entanto, nem sempre o cliente estará disponível pessoalmente, para isso é interessante possui uma ferramenta remota de mensagens (PRIKLADNICKI; WILLI; MILANI, 2014). Para manter a produtividade alta é preciso também ser simples quanto a arquitetura, modelagem e estilo de codificação, isso garante a facilidade na implementação. Porém, as soluções não podem perder qualidade, daí a maior dificuldade da metodologia, o *trade-off* entre tempo e qualidade. A coragem é necessária para mudar, inovar e aceitar uma opinião divergente que seja melhor, afinal, a XP preza pela abordagem realista e objetiva. Por se tratar de um ambiente colaborativo, o respeito é fundamental para que a equipe atinja os resultados. Criar oportunidades de aprendizado e colaboração valoriza os indivíduos e fortalece o comprometimento, superando as limitações individuais (PRIKLADNICKI; WILLI; MILANI, 2014).

2.1.4.3 Feature Driven Development

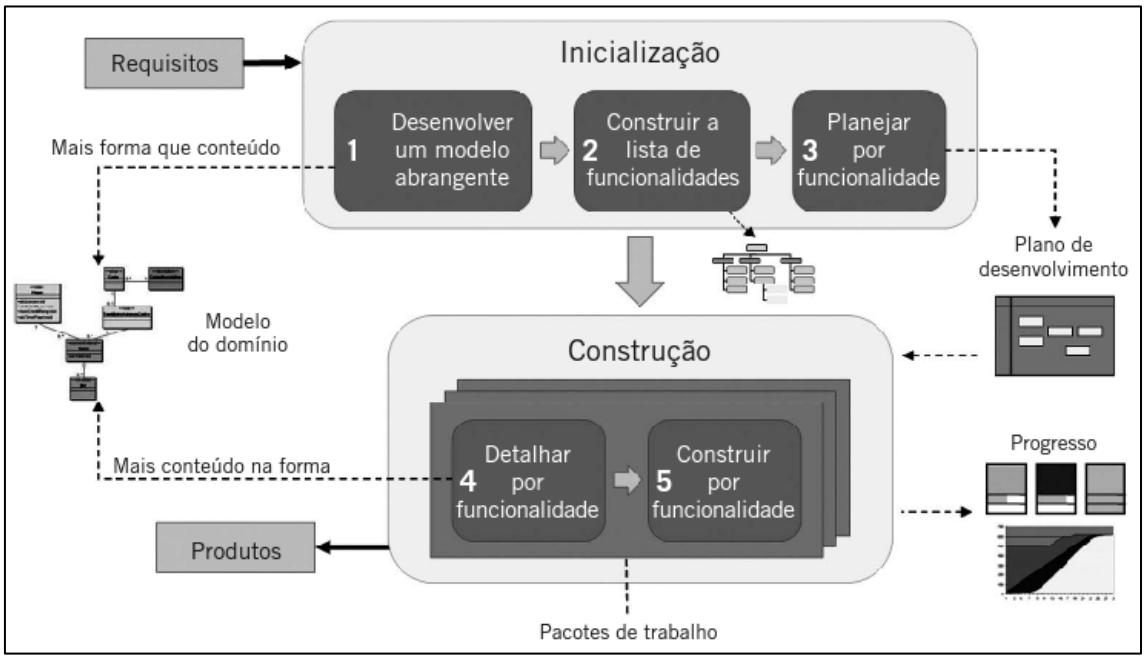
O *Feature Driven Development* (FDD) ou em português, Desenvolvimento Dirigido por Funcionalidades, é uma metodologia ágil para gerenciamento e desenvolvimento de software, a qual combina as melhores práticas do gerenciamento ágil de projetos com uma abordagem completa para Engenharia de Software orientada por objetos, conquistando os três principais públicos de um projeto de software: clientes, gerentes e desenvolvedores. No entanto, é preciso entender o que são funcionalidades. Funcionalidades são tarefas derivadas das diversas atividades de um processo de negócio que necessitam de auxílio de software para serem automatizadas (PRIKLADNICKI; WILLI; MILANI, 2014), em outras palavras, são os desejos do cliente traduzidos em ações que o *software* pode executar.

Segundo Szego (2004 apud PRIKLADNICKI; WILLI; MILANI, 2014), um dos criadores da FDD, as funcionalidades são enumeradas depois da atividade de modelagem inicial e são uma decomposição do domínio do problema, pois, sua matéria-prima são os requisitos do cliente, em qualquer forma disponível. Em geral, esses requisitos se encontram na mente das pessoas (conhecimento tácito), ou seja, não está documentado.

O FDD é composto por 5 processos distribuídos em 2 fases (iniciação e construção), a fase de iniciação aponta para o produto a ser construído com visão inicial de sua estrutura e suas funcionalidades, criando um plano inicial de entregas incrementais, enquanto que a fase de construção visa entregar os incrementos dos produtos de forma frequente, tangível e funcional, com qualidade para serem utilizados pelo cliente. A primeira fase (iteração zero) deve consumir até 4 semanas, e a segunda fase 2 semanas a cada iteração (PRIKLADNICKI; WILLI; MILANI, 2014). A Figura 19 apresenta o ciclo de vida do FDD, através de suas duas fases e seus cinco processos, descritos abaixo (RETAMAL, 2008):

- Desenvolver um modelo abrangente: realizar um estudo dirigido sobre o escopo do sistema e seu contexto;
- Construir a lista de funcionalidades: identificar todas as funcionalidades que satisfaçam os requisitos de maneira categorizada;
- Planejar por funcionalidade: planejar a ordem na qual as funcionalidades serão implementadas, baseada nas dependências entre elas, na carga de trabalho da equipe de desenvolvimento e também na complexidade das funcionalidades a serem implementadas;
- Detalhar por funcionalidade: produzir os pacotes de trabalho (conjunto de funcionalidades de mesma classe) através da produção dos diagramas de sequência para as funcionalidades, escrevendo-se os prefácios das classes e métodos;
- Construir por funcionalidade: produzir uma função com valor para o cliente, implementando os pacotes de trabalho. O código desenvolvido deve passar pelo teste de unidade e pela inspeção, para depois ser promovido à versão atual.

Figura 19 – Ciclo de Vida do FDD.



Fonte: PRIKLADNICKI; WILLI; MILANI, 2014, p. 75.

Diante do exposto, isto é, o referencial teórico acerca dos modelos de processo de *software*, da gestão de projetos de *software* e do desenvolvimento ágil de *software* faz-se uma reunião sucinta destes temas para encadeá-los ao objetivo e ao contexto do estudo.

2.2 CONSOLIDAÇÃO DO REFERENCIAL TEÓRICO

Sob o ponto de vista do presente trabalho, o referencial analisado permitiu estabelecer um marco conceitual a ser testado ao longo da pesquisa. Assim, o gerenciamento de *software* estabelece que é preciso planejar, organizar, monitorar e controlar os projetos de *software* a fim de garantir seu sucesso (PRESSMAN, 2006), o qual pode ser medido e avaliado a partir dos parâmetros de prazo, escopo e orçamento, além da satisfação do cliente (ARCHIBALD; PRADO, 2012).

É importante ressaltar que o modelo de processo de *software* (conjunto distinto de atividades, ações, tarefas e marcos que levam à produção de um produto de *software*) utilizado é, talvez, o grande e maior contribuinte para o sucesso ou fracasso de um projeto de *software*, pois este influencia diretamente no modo como serão conduzidos a comunicação, o planejamento, a modelagem, a construção e a implantação. Sendo assim é de extrema importância a escolha adequada do modelo a ser aplicado para o projeto de desenvolvimento de *software* em questão, pois cada projeto pode requerer um modelo diferente (PRESSMAN, 2006).

Portanto, o trabalho se relaciona à discussão sobre as diferenças entre as metodologias (teóricas e observada), de modo a correlacionar as possíveis deficiências encontradas com

os parâmetros de insucesso de projetos de *software* evidenciados por Archibald e Prado (2012) em organizações públicas de administração direta.

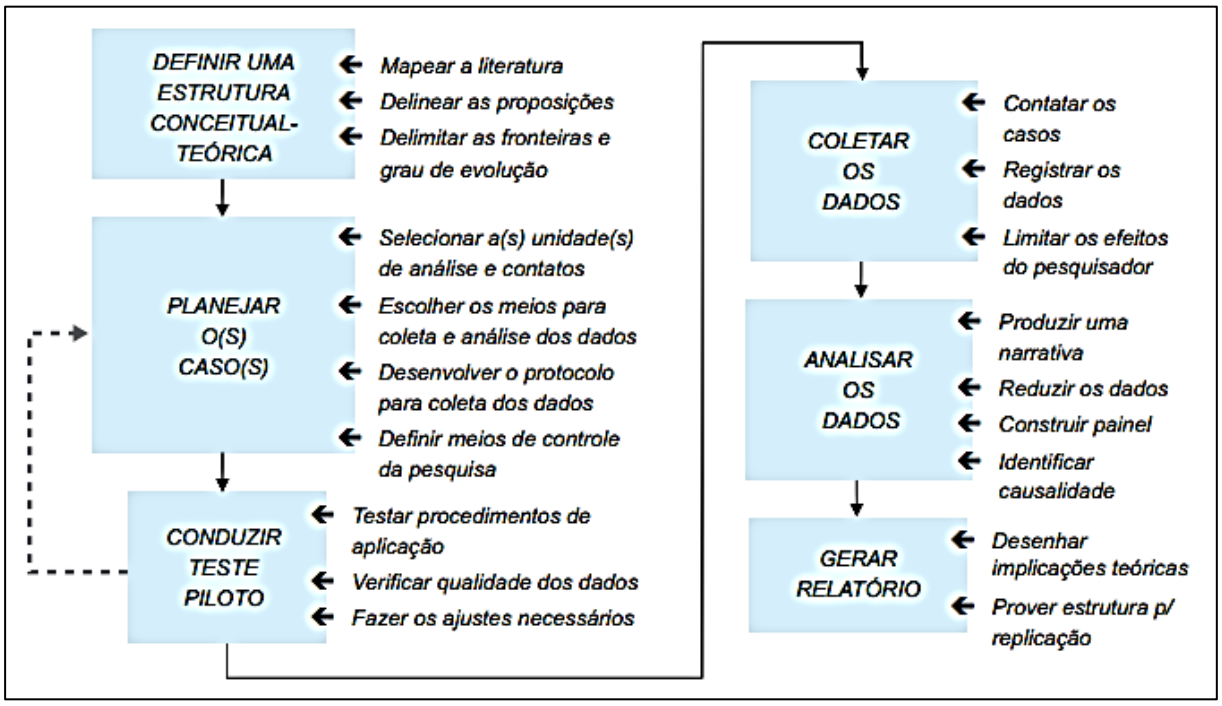
Vale ainda ressaltar que o sucesso de um projeto não depende somente da competência operacional, mas sim implica que as práticas de gerência de projetos estejam na cultura da empresa. Estas práticas devem ser reconhecidas e respaldadas pela alta administração. Este comprometimento gerencial irá estimular a gestão de projetos, além de permitir o fortalecimento dos valores básicos de um projeto: cooperação (trabalho de equipe), confiança e comunicações eficientes (PINTO; VASCONCELOS; LEZANA, 2014). Aliado a isso, existe o conceito de maturidade em gestão de projetos, o qual é definido como sendo o desenvolvimento de sistemas e processos, conjunto de atividades, de natureza repetitiva e que garantam alta probabilidade de sucesso (KERZNER, 2006 apud JUCÁ JUNIOR; CONFORTO; AMARAL, 2010). Isso significa atingir um patamar em que as práticas de gestão e controle dos projetos sejam institucionalizadas na organização, assim, a execução consistente deixa de depender unicamente da atitude dos profissionais (CRISSIS; KONRAD; SCHRUM, 2003 apud JUCÁ JUNIOR; CONFORTO; AMARAL, 2010). Contudo, não significa dizer que processos e sistemas repetitivos ou institucionalizados sejam garantia de sucesso, apenas aumentam sua probabilidade.

3. METODOLOGIA

Para a classificação do trabalho, utiliza-se critérios quanto à sua abordagem, sua natureza, seus objetivos e seus procedimentos. Quanto à abordagem a pesquisa será qualitativa, pois conforme Goldenberg (2009), a pesquisa não se preocupa com a representatividade numérica, mas, sim, com o aprofundamento da compreensão de um grupo social ou de uma organização. Assim sendo, o trabalho visa entender o modelo de gestão de desenvolvimento de *software* de uma instituição governamental de administração direta. A natureza da pesquisa será aplicada, pois gera conhecimentos novos de caráter aplicado e não em situação de laboratório ou teórica, e envolve verdades e interesses locais dirigidos à solução de problemas específicos (GERHARDT; SILVEIRA, 2009). Quanto aos objetivos a pesquisa será exploratória, pois visa investigar o modelo de gestão de desenvolvimento de *software* e compará-lo à literatura. Essas pesquisas envolvem: (a) levantamento da literatura da ciência afim; (b) entrevistas com pessoas que tiveram experiências práticas com o problema pesquisado; e (c) análise de exemplos que estimulem a compreensão (SELLTIZ, 1965). E, quanto aos procedimentos a pesquisa será conduzida por um estudo de caso, pois, consiste no estudo profundo e exaustivo de um ou poucos objetos, de maneira que permita seu amplo e detalhado conhecimento (GIL, 2002).

A Figura 20 apresenta as seis etapas do método de estudo de caso conforme Miguel (2007), o qual pressupõe que a aplicação do método seja realizada por um membro externo à organização, no caso, o próprio autor do trabalho.

Figura 20 – Condução do Estudo de Caso.



Fonte: MIGUEL, 2007, p. 221.

Etapa 1: Definição de uma estrutura Conceitual-Teórica

Nesta etapa, foram utilizados materiais como livros, teses e artigos para a fundamentação teórica-metodológica, de modo que os assuntos pesquisados são: Engenharia de *Software*, gerenciamento de projetos de *software* e metodologias ágeis. Para Miguel (2007), o referencial teórico também serve para delimitar as fronteiras do que será investigado, proporcionar o suporte teórico para a pesquisa (fundamentos) e também explicitar o grau de evolução (estado da arte) sobre o tema estudado, além de ser um indicativo da familiaridade e conhecimento do pesquisador sobre o assunto.

Etapa 2: Planejamento do(s) Caso(s)

Nesta etapa, o primeiro passo é a escolha da(s) unidade(s) de análise, ou seja, do(s) caso(s). Num primeiro momento deve ser determinada a quantidade de casos: único ou múltiplos casos (YIN, 2001 apud MIGUEL, 2007). Assim, definiu-se apenas um caso de estudo, o qual tratará do modelo de gestão de desenvolvimento de software de uma instituição governamental de administração direta através dos processos executados pelo Centro de Informática desta organização. A partir da seleção do caso, deve-se determinar os métodos e técnicas tanto para a coleta quanto para a análise dos dados. Para este fim, optou-se pela coleta de dados via entrevistas com os gestores e com os executores dos

processos de gestão de desenvolvimento de *software*. Uma vez escolhidas as técnicas para a coleta de dados, um protocolo deve ser desenvolvido. Basicamente, um protocolo deve considerar como partes relevantes:

> "... o contexto (área e local, unidade de análise, questões, procedimentos e fontes de informação), a parte a ser estudada (práticas, unidade de análise, questões, procedimentos e fontes de informação) e meios de controle da pesquisa (variáveis de controle e respectivas questões). " (SOUZA, 2005 apud MIGUEL, 2007. p. 223).

Portanto, este protocolo determina como objeto de estudo o Centro de Informática de um órgão público da área de educação. Neste ambiente, foi investigado o processo de Desenvolvimento Sistemas através de entrevistas de duração de 1 hora com a gerência de desenvolvimento de sistemas, com o chefe da área temática de desenvolvimento e com a equipe de desenvolvimento, atingindo assim o proposto, conseguir reunir-se com os gestores e executores do processo. Os procedimentos e variáveis de controle utilizados seguiram o exposto pelo Quadro 4, de forma que essa programação decorre de uma adaptação da sistemática utilizada pelo projeto MAPROEx, o qual o autor deste trabalho faz parte. Este projeto decorre de um Termo de Execução Descentralizada (TED 14-150-00) entre o Exército Brasileiro e a Fundação Universidade de Brasília (UnB) / Departamento de Engenharia de Produção (EPR) / Centro Interdisciplinar de Estudos em Transporte (CEFTRU que teve início em setembro de 2014 e se encerrará em fevereiro de 2017, além de contar com uma equipe composta por 50 colaboradores, sendo eles: 5 professores doutores, 21 colaboradores (graduados, mestrandos e mestres) e 24 estagiários de Engenharia de Produção e de Engenharia de Software; assim sendo, é certo dizer que sua sistemática é consolidada.

Quadro 3 – Programação para Mapeamento de Processos.

DIA 1	DIA 2	DIA 3	DIA 4	DIA 5	DIA 6	DIA 7	DIA 8
							
Realizar entrevista 1	Elaborar diagrama	Validar diagrama com Orientador	**Realizar entrevista 2**	Revisar diagrama	Validar diagrama com Escritório	**Validar diagrama com Gestor**	Concluir diagrama

Fonte: Projeto MAPROEx (adaptado).

Portanto, o presente trabalho realizou 2 reuniões de entrevistas com o cliente e 3 validações (uma com o orientador doutor, outra com o escritório de processos e outra com o próprio cliente) para elaborar o diagrama do processo em questão.

Etapa 3: Condução de um Teste Piloto

Apesar de incomum em estudos de casos, é sempre importante a condução de um teste piloto pelo pesquisador antes de partir para a coleta de dados (MIGUEL, 2007). O teste propicia a verificação dos procedimentos de aplicação com base no protocolo, visando seu aprimoramento. O teste piloto é importante, pois é realizado preliminarmente em uma escala menor de abrangência, ou seja, menor número de entrevistas, que servirá como orientação para a realização da pesquisa propriamente dita, uma vez que fornecerá as devidas correções ao questionário. Contudo, os procedimentos utilizados para a coleta de dados foram baseados na sistemática utilizada pelo projeto supracitado, portanto, considerou-se desnecessária a realização deste exame.

Etapa 4: Coleta dos Dados

Assim como já mencionado, a coleta dos dados segue a metodologia do Projeto MAPROEx. Com base nisso, serão coletados os dados do processo de Desenvolvimento de Sistemas da organização estudada. Vale ressaltar a importância de tentar limitar os efeitos do próprio pesquisador, que deve sempre ter em mente que ele é um elemento estranho no contexto analisado; portanto, em termos de efeitos do pesquisador no caso, ele não pode influenciar os respondentes (SOUZA, 2005 apud MIGUEL, 2007).

Etapa 5: Análise dos Dados

A partir do conjunto de dados coletados e as múltiplas fontes de evidência, o pesquisador deve produzir uma narrativa geral do caso (seja uma dissertação, tese, relatório de pesquisa ou artigo), o que não quer dizer que toda a coleta de dados será incluída no relatório da pesquisa, de modo que seja incluído na análise somente aquilo que é essencial e que tem estreita ligação com os objetivos da pesquisa (MIGUEL, 2007). Assim, será elaborado um diagrama com padrão BPMN para o caso observado, posteriormente um quadro comparativo das atividades descritas pela literatura com as atividades percebidas pelo mapeamento do processo.

Etapa 6: Geração do Relatório da Pesquisa

Concluídas as etapas anteriores, deve-se sintetizá-las em um relatório de pesquisa. Os resultados devem estar estreitamente relacionados à teoria, isto é, não se deve ajustar a teoria aos resultados e evidências, mas sim, os resultados e as evidências que deverão ser associados à teoria, possibilitando, inclusive, a geração de uma nova teoria (MIGUEL, 2007). Portanto, será elaborado o relatório, e este será confrontado com a teoria pesquisada, dando origem a diversos quadros consolidados que apresentem as semelhanças e diferenças entre a literatura e o caso observado.

4. ESTUDO DE CASO

Antes de dar início às atividades de modelagem, análise e desenho do processo, é essencial conhecer a organização, seu contexto, sua missão e objetivos estratégicos. Trata-se da fase de preparo.

4.1 PREPARAÇÃO PARA ANÁLISE DO PROCESSO

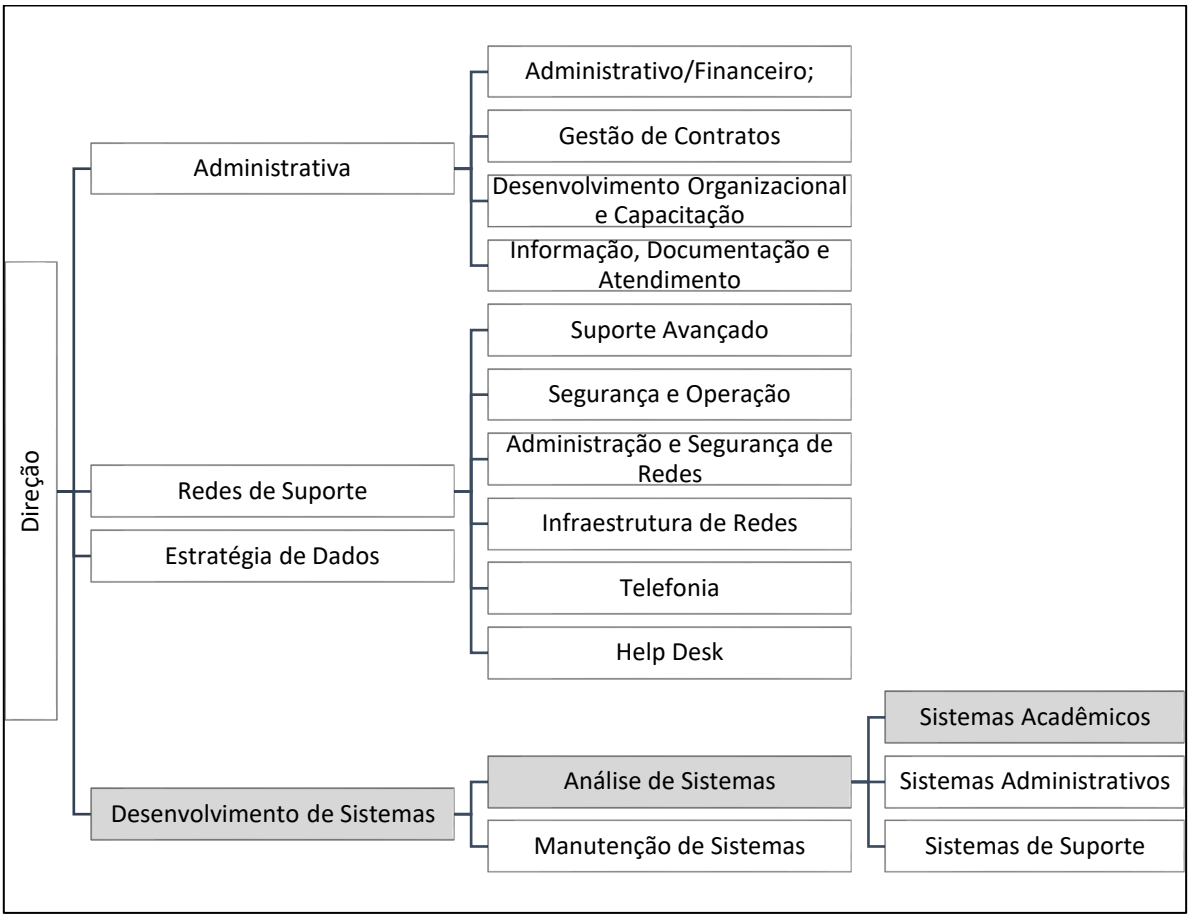
O objeto de estudo em foco neste trabalho está inserido dentro da estrutura organizacional extensa e complexa, típica de uma grande Instituição de Ensino Superior (IES) pública brasileira. Localizada no Distrito Federal, a IES possui diversos *campi* espalhados pelo quadrilátero. No *campus* principal encontra-se o Centro de Informática – CPD, que tem a intenção de desenvolver as atividades de caráter permanente de apoio, necessárias ao desenvolvimento do ensino, da pesquisa e da extensão no que se refere ao processamento de dados. Apesar de tratar-se de um Centro de Informática a sigla CPD permanece para evitar enganos com a sigla do Conselho de Informática que fora criado primeiro.

Em 1996, o CPD foi alçado à condição de Unidade Gestora, com delegação e competência para, por meio de seu Diretor, praticar os atos de gestão orçamentária, financeira e patrimonial, observadas as normas que regem a execução orçamentária do Governo Federal, podendo também celebrar contratos e convênios necessários à implementação de suas atividades.

Assim, o CPD desenvolve e implementa a maior parte de todos os sistemas corporativos que são necessários aos mais diversos setores da universidade. Foi pioneiro no desenvolvimento de sistemas de recuperação de informações em tempo real através de terminais. E o primeiro a desenvolver um sistema de informatização para uma biblioteca universitária no país, tendo um sistema que pesquisa e atualiza o banco de dados em tempo real.

O Centro de Informática é um Órgão Complementar da universidade, responsável pela Tecnologia da Informação e é subordinado diretamente à Vice-Reitoria (VRT). Enquanto que o organograma do CPD é estruturado em 3 níveis hierárquicos: a direção, as gerências (administrativa, de redes e suporte, de estratégia de dados e de desenvolvimento de sistemas) e as subáreas, conforme mostrado pela Figura 21. A Gerência de Desenvolvimento de Sistemas, que é o caso do objeto do estudo, tem como competência o atendimento às demandas para desenvolvimento de sistemas, podendo ser externa ou interna ao CPD, mas sempre interna à Universidade. A ela, estão vinculadas as subáreas de Análise de Sistemas e Manutenção de Sistemas, onde a primeira deve receber e analisar as demandas de soluções de software de caráter institucionais enquadradas como adaptativa, perfectiva, evolutiva ou projetos novos, enquanto que a segunda, deve proceder a manutenção corretiva, adaptativa ou evolutiva nos sistemas corporativos em produção.

Figura 21 – Organograma do Centro de Informática.



Fonte: CPD (adaptado).

O objetivo estratégico geral do CPD pode ser resumido em: viabilizar soluções de tecnologia da informação que promovam a disponibilidade, integridade, confiabilidade e autenticidade das informações dos ativos relacionados aos sistemas informatizados da Universidade de Brasília. De modo que o CPD busca a realização desse objetivo principal através dos seguintes objetivos específicos:

- Promover e incentivar a informática na Universidade de Brasília visando obter eficiência institucional em todos os níveis;
- Promover e incentivar a informática na Universidade de Brasília para alcançar maior eficácia no suporte às atividades de ensino, pesquisa, extensão e administração da Instituição;
- Promover meios para o compartilhamento de recursos computacionais entre a comunidade acadêmica da UnB e as redes de pesquisa nacionais e internacionais;
- Desenvolver, implantar e manter sistemas em mainframe e em microcomputadores;
- Supervisionar, coordenar e controlar as atividades relacionadas com pesquisa, desenvolvimento e manutenção de hardware, software e rede de teleprocessamento, assim como as relacionadas com a manutenção ambiental e operação de computadores;

- Planejar e coordenar a execução de serviços relacionados com o tratamento eletrônico de informações.

O objetivo estratégico geral e os objetivos específicos podem perfeitamente ser compreendidos, respectivamente, como macroprocesso e processos do CPD. Cada um desses processos é composto por subprocessos, atividades e tarefas. Porém, há um processo essencial sem o qual a viabilização de todos os demais processos e objetivos estratégicos fica impedida ou prejudicada: o processo de desenvolvimento de sistemas. Quaisquer melhorias propostas nesse processo têm o potencial de proporcionar diversos impactos positivos em todo o CPD, gerando agregação de valor imediatamente. Trata-se, portanto, de um processo primário e crítico.

A partir dessa contextualização, o processo Desenvolvimento de Sistemas do Centro de Informática foi então analisado e descrito na forma de um modelo.

4.2 ANÁLISE DO PROCESSO (MODELO *AS IS*)

Para facilitar a compreensão do modelo existente (*as is*), o mesmo será apresentado detalhando cada uma de suas fases (anexo D).

O modelo observado possui 79 atividades que são desenvolvidas através da interação de 5 atores: o cliente que demanda o projeto de desenvolvimento de sistemas; a direção do Centro de Informática que recebe essas demandas e negocia com o cliente a possibilidade de executar o projeto; a gerência de desenvolvimento de sistemas que interage mais com o cliente e é a responsável pela entrega efetiva; o chefe da área temática de desenvolvimento de sistema que é o responsável pelo desenvolvimento; e a equipe de desenvolvimento, composta por desenvolvedores, analistas e técnicos que de fato irão desenvolver o sistema. Vale lembrar que o desenvolvimento desses sistemas não é fornecido para fora da instituição pública em questão, isto é, o Centro de Informática somente atende às exigências deste órgão púbico, portanto, o cliente é um cliente interno da instituição, seja ele do próprio CPD ou de qualquer outra área da IES.

Após o modelo ter sido mapeado, ele foi analisado, identificado e classificado de acordo com as cinco fases dos processos de desenvolvimento de *software* de Pressman (2006): comunicação, planejamento, modelagem, construção e implantação; e ainda uma sexta fase foi colocada, a etapa de encerramento.

4.2.1 COMUNICAÇÃO

O processo se inicia mediante a identificação por parte do cliente de uma necessidade de informática, ou seja, uma demanda reprimida de sistema. Esta demanda é disparada ao Centro de Informática em forma de Ordem de Serviço via um sistema de gerenciamento de serviços de TI (CITSmart ITSM), o centro o recebe através da Direção, a qual irá julgar a

pertinência deste pedido, podendo negá-lo ou atribuí-lo à Gerência de Desenvolvimento de Sistemas, que irá marcar uma reunião com o cliente a fim de entender melhor o contexto deste projeto e avaliar se o Centro tem capacidade para desenvolvê-lo. Após este crivo, o projeto será encaminhado à Chefia da Área Temática de Desenvolvimento competente para desenvolver o projeto, haja visto que existem três áreas (sistemas administrativos, acadêmicos e de suporte), onde o Chefe da Área Temática de Desenvolvimento irá marcar uma outra reunião com o cliente para coletar os requisitos do projeto, ao passo que também avaliará a possibilidade de atender aos requisitos levantados assim como o cliente deseja. Em caso de aprovação, a ata desta reunião com os requisitos levantados é encaminhada à Equipe de Desenvolvimento que irá elaborar o Documento de Abertura do Projeto (anexo E). Este documento será avaliado tanto pelo Chefe da Área Temática de Desenvolvimento, quanto pela Gerência de Desenvolvimento de Sistemas que irá apresentá-lo ao cliente para sua assinatura, configurando-se assim o início do projeto e o término da etapa de comunicação.

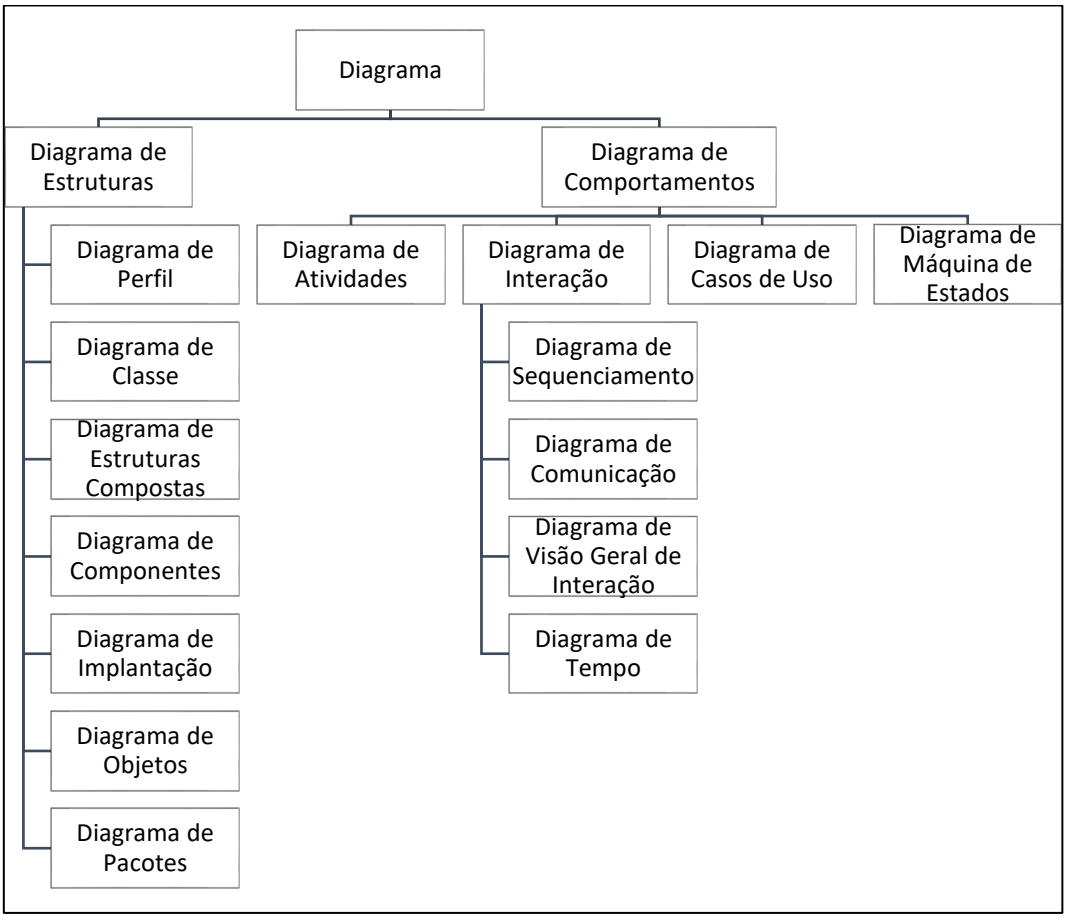
4.2.2 PLANEJAMENTO

Após a autorização do início do projeto, realiza-se a análise dos recursos humanos e tecnológicos disponíveis e necessários. Com esta análise, é elaborado um cronograma e designada uma Equipe de Desenvolvimento responsável pela execução do projeto. Esta equipe é sempre formada por no mínimo 3 integrantes: um desenvolvedor, um analista e um técnico; podendo ser composta por mais membros, a depender da disponibilidade de recursos humanos. A equipe irá, nesta etapa, analisar os produtos que devem ser entregues: tecnologia, plataforma e banco de dados, os quais serão consolidados num Plano de Projeto que subsidiará a etapa a seguir de modelagem.

4.2.3 MODELAGEM

Concluído o planejamento, a Equipe de Desenvolvimento irá modelar uma proposta de solução com base no Plano de Projeto inicialmente pensado. Neste momento serão modelados os diagramas de estruturas e comportamentos (Figura 22) para que o sistema possa ser construído. Essa modelagem se baseia no padrão UML, porém não faz uso de todos os diagramas previstos. A modelagem somente se encerra após a autorização do Chefe da Área Temática de Desenvolvimento, isto é, após o seu aval sobre a Proposta de Solução (modelagem dos diagramas), conferindo se está adequada ao projeto de desenvolvimento de sistema.

Figura 22 – Diagrama de Estrutura e Comportamento da UML.



Fonte: OMG, 2001b (adaptado).

4.2.4 CONSTRUÇÃO

Em seguida inicia-se a etapa de construção, onde serão escritos os códigos das funcionalidades do sistema. Neste ponto cabe à própria Equipe de Desenvolvimento definir a regra de sequenciamento do desenvolvimento, tendo assim autonomia para priorizar o ritmo de trabalho. A princípio, cada Equipe de Desenvolvimento está alocada em apenas um projeto, portanto, a priorização do ritmo de trabalho refere-se ao fato de definir a regra de prioridades da construção das funcionalidades: quais são mais complexas, quais são mais importantes, quais devem ser desenvolvidas primeiro. Após a conclusão de uma funcionalidade, a própria equipe realiza teses de tela, a fim de identificar possíveis falhas do sistema, podendo assim corrigi-las antes de apresentar ao Cliente. Vale destacar que a cada interação ocorre uma validação da seguinte maneira: sempre que uma funcionalidade é concluída e aprovada nos testes de tela, independentemente do tempo que ela tenha demorado para ser construída, esta é apresentada ao Chefe da Área Temática de Desenvolvimento que a exibe ao Cliente para sua avaliação e retorno de *feedback*, isto permite a Equipe de Desenvolvimento rever a funcionalidade e ajustá-la ainda nesta fase, a qual somente se encerra quando todas as funcionalidades/módulos do sistema estiverem sido construídos.

Essa etapa requer uma comunicação constante e efetiva com o Cliente, portanto, a proximidade e disponibilidade do Cliente são fundamentais para o sucesso desta etapa, do contrário, o cronograma pode sofrer atrasos. O cumprimento do cronograma, bem como a supervisão da Equipe de Desenvolvimento é de incumbência do Chefe da Área Temática de Desenvolvimento, logo, ele deve estar atento ao ritmo de trabalho de sua equipe e ao contato com o Cliente.

4.2.5 IMPLANTAÇÃO

Com a conclusão da fase de construção, inicia-se a etapa de implantação. Nesta etapa o projeto será apresentado à Gerência de Desenvolvimento de Sistemas, que pode solicitar ajustes e/ou refinamentos ao produto, pois estará verificando se este atingiu os objetivos de entrega estabelecidos no Documento de Abertura do Projeto. Após aprovado, o projeto será apresentado ao Cliente que também irá avaliá-lo e se necessário, retornará com *feedback*. Ainda que o Cliente tenha aprovado cada uma das funcionalidades ao longo da fase de construção do sistema, ao visualizá-lo por completo, pode-se ter uma visão diferente, fazendo assim, novas solicitações de mudança. No entanto, essas mudanças ao final do projeto, em geral, acarretam num estouro do custo e do tempo, daí a importância de a Gerência de Desenvolvimento de Sistemas confrontar os desejos do Cliente com o que ele contratou no início através do Documento de Abertura do Projeto.

Posteriormente à aprovação do Cliente, a Equipe de Desenvolvimento inicia a elaboração de manuais e o planejamento de treinamentos. Esses manuais são disponibilizados virtualmente para consulta e servem como guias que irão auxiliar o usuário a utilizar o sistema, seja fornecendo orientações para o uso ou instruções para soluções de problemas. Neste momento, o sistema é disponibilizado ao Cliente em um ambiente virtual controlado (*link* externo) e em condições de uso, assim a Equipe de Desenvolvimento pode acompanhar a utilização do sistema, podendo ainda fazer alguma modificação.

4.2.6 ENCERRAMENTO

Com o fim da etapa de implantação, a Equipe de Desenvolvimento elabora o Documento de Encerramento do Projeto, o qual irá para aprovação da Gerência de Desenvolvimento de Sistemas, e, posteriormente para assinatura do Cliente, configurando desta forma a aceitação do sistema desenvolvido. Em seguida, o ambiente virtual deixa de ser controlado, isto é, torna-se totalmente utilizável e disponível para o Cliente, encerrando o acesso do CPD ao sistema no que se refere ao desenvolvimento. A Ordem de Serviço é encerrada pela Direção através do sistema de gerenciamento de serviços de TI (CITSmart ITSM) e o processo é finalizado.

O processo de produção (incremento de software) não foi mapeado, pois não era alvo deste trabalho.

4.3 COMPARAÇÃO DO MODELO OBSERVADO COM A LITERATURA

Uma vez que o processo foi mapeado, refletindo a forma de funcionamento vigente, torna-se mais fácil a percepção de problemas e pontos críticos que interferem na eficiência do processo, na geração de valor e no alcance dos objetivos.

4.3.1 MODELO DE DESENVOLVIMENTO

O modelo mapeado caracteriza-se pela semelhança com o modelo em cascata, pois segue um sequenciamento sistemático para o desenvolvimento de *software*, até que a versão final e executável do sistema seja disponibilizada para o cliente ao final do processo, porém ele também é iterativo, assemelhando-se ao modelo em espiral, pois apresenta *loops* de verificações e validações junto ao cliente a cada funcionalidade desenvolvida, mesmo que ainda não seja executável. No entanto, nem todas as etapas da espiral, realmente seguem uma espiral. Portanto, pode-se dizer que o modelo observado presume uma cascata em espiral.

O Quadro 4 exibe um comparativo das atividades intrinsecamente pertencentes aos processos de software de Pressman (2006) (cascata e espiral) com o que foi observado através do mapeamento. A primeira coluna refere-se às etapas do processo de *software* de Pressman (2006), a segunda coluna descreve as atividades de cada uma dessas etapas, a terceira coluna refere-se à aderência ao modelo em cascata, isto é, se a atividade em questão foi observada ou não, e a última coluna diz respeito à adesão ao modelo em espiral, ressaltando se a atividade em relacionada é iterativa ou não.

Quadro 4 – Comparativo entre o Modelo em Cascata de Pressman com o Modelo de Desenvolvimento de *Software* Observado.

ETAPA	ATIVIDADES DE PROCESSOS DE SOFTWARE	ADERÊNCIA AO MODELO EM CASCATA	ADERÊNCIA AO MODELO EM ESPIRAL
Comunicação	Identificar os requisitos com o cliente.	Sim	Sim
	Definir o escopo do sistema.	Sim	Não
Planejamento	Avaliar a viabilidade do projeto.	Sim	Não
	Descrever as tarefas.	Não	Não
	Descrever os riscos.	Não	Não
	Descrever os recursos.	Sim	Não
	Descrever os produtos.	Sim	Não
	Elaborar o cronograma.	Sim	Sim
	Estimar o custo do projeto.	N.A.	N.A.
Modelagem	Conceber dos modelos.	Sim	Sim
Construção	Gerar os códigos.	Sim	Sim
	Realizar os testes para identificação de erros.	Sim	Sim
	Realizar os testes de aceitação.	Sim	Sim

continua

continuação

ETAPA	ATIVIDADES DE PROCESSOS DE SOFTWARE	ADERÊNCIA AO MODELO EM CASCATA	ADERÊNCIA AO MODELO EM ESPIRAL
Implantação	Elaborar os manuais, guias e procedimentos de instalação.	Sim	Não
	Entregar o software ao cliente.	Sim	Sim
	Avaliar o feedback do cliente em relação a defeitos e/ou modificações.	Sim	Sim

Fonte: o autor.

Através do Quadro 4 é possível perceber que o modelo observado realiza 13 das 16 atividades do modelo em cascata, enquanto que a iteração só ocorre em 8 das 16 atividades do modelo em espiral. Esse fato, corrobora para dizer que o modelo pode ser descrito por uma cascata em espiral, pois é parcialmente semelhante ao modelo em cascata e parcialmente semelhante ao modelo em espiral.

Contudo, não é correto afirmar que a eficiência no desenvolvimento de *software* será atingida com o maior grau de aderência aos modelos pré-existentes, pois essa premissa não é válida.

4.3.2 GESTÃO DE PROJETOS

A gestão de projetos visa gerenciar o pessoal, o produto e o processo a fim de alcançar o sucesso no desenvolvimento de *software*, para isso é necessário atentar-se aos seguintes princípios básicos: métricas, estimativa de projetos, cronogramação, gestão de riscos, gestão da qualidade e gestão das modificações (PRESSMAN, 2006). O Quadro 5 exibe a comparação desses princípios básicos de gestão de projetos com o modelo observado.

Quadro 5 – Princípios Básicos de Gestão de Projetos Observados.

PRINCÍPIOS BÁSICOS	OBSERVAÇÃO DO OBJETO DE ESTUDO
Métricas	Coleta apenas as métricas de processo. Desde 2010, 9 sistemas dos 23 iniciados foram concluídos.
Estimativas de projetos	Viabilidade tecnológica: analisa a capacidade de entregar a tecnologia, a plataforma e o banco de dados; Viabilidade financeira: realizada somente quando há necessidade de aquisição de alguma ferramenta de software; Viabilidade de tempo: é menos valorizada do que a qualidade e não há concorrência para o serviço prestado; Viabilidade de recursos: focada principalmente na necessidade de material humano. O ideal é que uma equipe de desenvolvimento trabalhasse apenas em um projeto por vez, porém isso não ocorre, devido à falta de mão de obra.
Cronogramação	Realiza completamente a etapa de definição de marcos de referência; Parcialmente: compartimentalização, interdependência, definição de responsabilidades e definição de resultados; e Não realiza: atribuição de tempo, validação de esforço.

continua

continuação

PRINCÍPIOS BÁSICOS	OBSERVAÇÃO DO OBJETO DE ESTUDO
Gestão de risco	Estratégia reativa, não há nenhuma identificação dos riscos durante o planejamento do projeto.
Gestão da qualidade	Depende das métricas de projeto, as quais não são coletadas. Portanto, a qualidade pode, apenas, ser acompanhada, mas não gerenciada.
Gestão das modificações	Realizada através do sistema de gerenciamento de serviços de TI CITSmart ITSM por meio da funcionalidade 'Gerência de Mudanças'.

Fonte: o autor.

Em relação às métricas, são coletadas apenas as métricas de processo, e não as de projeto. Isso permite à organização ter ideia da eficácia do processo de desenvolvimento de *software*, pois estes, são dados coletadas ao longo de todos os projetos e durante longos períodos, no entanto, não é possível ter a visão do andamento individual de cada projeto. Desde 2010, a área temática de desenvolvimento entrevistada entregou 9 sistemas (MatrículaWEB, MençãoWEB, SIBOL, SIBOLWEB, SIDIP, SIEX, SIGRA, SIPPOS e Sistemas de Avaliação de Disciplinas) dos 23 iniciados a IES do objeto de estudo. Os outros, foram descontinuados e/ou estão em andamento.

No que se refere às estimativas de projetos de *software*, elas devem estar relacionadas a viabilidade tecnológica, financeira, de tempo e de recursos (PUTNAM & MYERS, 1997 apud PRESSMAN, 2006). O objeto de estudo analisa a viabilidade tecnológica quanto a sua capacidade de entregar a tecnologia, a plataforma e o banco de dados tal qual o cliente deseja. A avaliação financeira, somente é realizada quando a viabilidade tecnológica apontada pela equipe de desenvolvimento sinaliza a necessidade de aquisição de alguma ferramenta de *software* para o desenvolvimento do projeto, no entanto, nesses casos, compete à direção julgar a continuidade desse projeto, pois o recurso deve ser descentralizado junto à reitoria, implicando dessa forma, num entrave burocrático. A viabilidade quanto ao tempo não é analisada de forma tão criteriosa, primeiramente, porque não há concorrência para o serviço prestado, e em segundo lugar, porque a qualidade é mais valorizada, tendo em vista que a instituição é perene. Por fim, a viabilidade de recursos observa a necessidade de material humano para o desenvolvimento dos projetos, haja visto que o ideal é que se tivesse ao menos um desenvolvedor, um analista e um técnico alocado em cada projeto, e que eles somente trabalhassem em um projeto por vez, porém isso não é ocorre, devido à falta de mão de obra.

Com relação a cronogramação, o objeto de estudo realiza apenas um dos sete princípios básicos citados por Pressman (2006) de maneira completa, que é a definição de marcos de referência; de maneira parcial, são executados os seguintes princípios: compartimentalização, interdependência, definição de responsabilidades e definição de

resultados; pois não se chega ao nível de análise das tarefas, apenas das atividades. Os outros princípios não são considerados.

Quanto a gestão do risco, as estratégias são sempre reativas, não há nenhuma identificação dos riscos durante o planejamento do projeto. À medida que algum problema surge, busca-se solucioná-lo baseado nas experiências passadas, porém isso também não é documentado.

No que diz respeito à gestão da qualidade, ela depende das métricas de projeto, e como visto anteriormente, estas métricas não são coletadas. Portanto, a qualidade do projeto de *software* estudado pode, apenas, ser controlada, mas não gerenciada. Pois, o controle da qualidade envolve inspeções, revisões e testes ao longo do projeto de *software* (PRESSMAN, 2006). Esse controle ocorre por meio das validações das funcionalidades de *software* que ocorrem junto ao cliente, por intermédio dos testes de tela do sistema realizados pela equipe de desenvolvimento, e através das revisões do sistema após os *feedbacks* do cliente.

Considerando a gestão das modificações, as mudanças podem ocorrer devido a novas condições de negócio ou de mercado que geram novos requisitos ou novas regras de negócio; novas necessidades do cliente que exigem novos dados, funcionalidades ou serviços; reorganização do negócio que causa alterações nas prioridades do projeto ou na estrutura da equipe; restrições de orçamento ou cronograma que causam redefinição do sistema ou do produto (PRESSMAN, 2006). A organização estudada realiza a gestão das modificações através do sistema de gerenciamento de serviços de TI CITSmart ITSM, onde são também abertas as Ordens de Serviço de desenvolvimento. No CITSmart ITSM existe uma funcionalidade denominada: Gerência de Mudanças, a qual permite gerenciar os SCIs até o nível da camada de Controle de Versão do modelo de Camadas do Processo de SCM (Figura 17). Porém, esse processo é burocrático, pois todas as modificações devem ser acompanhadas e revisadas antes de serem aprovadas.

4.3.3 DESENVOLVIMENTO ÁGIL

Antes de apresentar os resultados obtidos acerca do desenvolvimento ágil é importante citar que a organização observada diz utilizar métodos deste tipo, sobretudo o modelo SCRUM. Contudo, o presente trabalho buscou identificar as minúcias ágeis executadas pelo processo independente do modelo escolástico ao qual ele pertence. Sendo assim, o ideal seria que fossem utilizadas as melhores práticas estabelecidas pelas metodologias ágeis.

O trabalho observou que a organização estudada de fato utiliza diferentes práticas das metodologias SCRUM, XP e FDD, no entanto, isso não quer dizer que utiliza as melhores de cada uma delas, tampouco utiliza de maneira adequada.

Quanto a metodologia SCRUM, que pressupõe equipes ágeis de pequeno porte, multidisciplinares, auto organizados e com foco na melhoria contínua, além das capacidades de transparência, inspeção e adaptação (AUDY, 2015; SCHWABER; SUTHERLAND, 2013), a organização estudada segue a metodologia de maneira parcial, pois os requisitos não são bem definidos, o *Scrum Team* não desempenha as mesmas atribuições definidas pela metodologia e os eventos estão atrelados à funcionalidade/módulo do sistema, e não ao tempo, o qual deveria ser fixo. Essas podem ser acompanhadas pelo Quadro 6 abaixo.

Quadro 6 – Práticas Observadas do Modelo SCRUM.

PRÁTICAS DO SCRUM		OBSERVAÇÃO DO OBJETO DE ESTUDO
Requisitos	*Product Backlog*	Existe, inicialmente, na forma do documento Proposta de Solução, porém, apesar de evoluir com o produto e o ambiente, ele não é atualizado.
	Sprint	Não é bem definido, pois os intervalos de entrega não são periódicos, baseiam-se na entrega e não no tempo.
	Sprint Backlog	Não é bem definido, pois não segue uma ordem específica e tal atividade é realizada pela Equipe de Desenvolvimento
Scrum Team	*Product Owner*	Existe na figura do Chefe da Área Temática de Desenvolvimento, porém não define a ordem com que os itens irão ser executados durante o *Sprint Backlog*.
	Time de Desenvolvimento	Existe na figura da Equipe de Desenvolvimento, e, de fato, é autogerenciável e tem autonomia de escolher a melhor forma de realizar suas atividades.
	Scrum Master	Deveria existir na figura do da Gerência de Desenvolvimento de Sistemas, porém não garante de forma completa o entendimento da teoria, práticas, regras e interações do SCRUM.
Eventos	*Sprint Planning*	Ocorre somente entre a Equipe de Desenvolvimento, onde definem as funcionalidades que serão desenvolvidas.
Eventos	*Daily Scrum*	Ocorre exatamente como previsto: reuniões diárias e curtas (5 minutos em média), onde busca-se responder o que foi feito, o se pretende fazer e o que está te impedindo de fazer algo, caso haja.
	Sprint Review	Ocorre exatamente como previsto: apresentação das funcionalidades desenvolvidas durante a *Sprint* e inspeção do produto que gera possíveis adaptações para as próximas *Sprints*.
	Sprint Retrospective	Ocorre de maneira parcial, pois nem sempre acontece imediatamente após a *Sprint Review* ou nem acontece.

Fonte: o autor.

Assim, compreende-se que não é correto afirmar que a organização pratica o SCRUM, pois uma vez que o tempo das *sprints* não é fixo, o modelo perde o caráter ágil. O correto seria tratar o escopo como variável e não a dimensão temporal.

Em relação a metodologia XP, a qual mira a alta qualidade e a alta produtividade por meio de valores e princípios sinergicamente envolvidos na instituição, percebeu-se que a instituição estudada cultiva muito destes valores e princípios, no entanto, a sinergia entre eles não é tão marcante a ponto de os pontos fortes sobrepujarem os pontos fracos assim como descrito por Prikladnicki, Willi e Milani (2014).

O Quadro 7 salienta sobre as principais práticas do modelo XP, de modo a relatar como elas são executadas pela instituição estudada. A comunicação constante com o cliente não ocorre durante a etapa de construção, quando o desenvolvedor está com alguma dúvida, ele

busca resolvê-la olhando os requisitos levantados e conversando com a própria equipe de desenvolvimento, assumindo uma resposta como certa, ou seja, nesse momento não há comunicação com o cliente, porém há validações a cada término de funcionalidade, o que permite esclarecer as dúvidas antes que todo o sistema esteja pronto, caso em que o retrabalho tenderia a ser maior. O desenvolvimento de uma ferramenta remota de comunicação com o cliente acarretaria em grandes benefícios de produtividade. A arquitetura, a modelagem e a codificação não são padronizadas, o que dificulta a celeridade do desenvolvimento do sistema, ora pode ser simples, ora não. Os produtos, em geral, possuem uma boa qualidade e aceitação, pois são desenvolvidos e validados parcialmente ao longo de todo o processo, de forma a manter o escopo. Porém, o produto quase nunca é entregue na data que havia sido estabelecida no início do projeto, devido ao baixo efetivo da equipe de desenvolvimento e o acúmulo de atrasos decorrentes das comunicações com o cliente, que pode demorar a agendar as validações de funcionalidades.

Quadro 7 – Práticas Observadas do Modelo XP.

PRINCIPAIS PRÁTICAS DO MODELO XP	OBSERVAÇÃO DO OBJETO DE ESTUDO
Comunicação constante durante a etapa de construção	Eficiente quanto às validações das funcionalidades, porém não ocorre durante as atividades de codificação.
Ferramenta remota de comunicação com o cliente	Não possui.
Simplicidade da arquitetura, modelagem e codificação	Não possui um padrão formal (documentado) bem estabelecido.
Trade-off entre tempo e escopo	Mantêm o escopo, porém excede o cronograma recorrentemente.

Fonte: o autor.

Portanto, também não é correto afirmar que o modelo XP é seguido pela organização estudada, pois além de não seguir as principais práticas apontadas pelo Quadro 7, não foi possível constatar os valores e princípios que a metodologia preconiza. Em parte, isso se deve ao fato de o observador não possuir o domínio e conhecimento completo das práticas XP que lhe confeririam capacidade para identificá-las.

No tocante a metodologia FDD, a qual divide o desenvolvimento de *softwares* em duas fases, uma que cria um plano inicial de entregas incrementais e a outra que, de fato, entrega os incrementos dos produtos de maneira frequente, tangível e funcional (PRIKLADNICKI; WILLI; MILANI, 2014), a instituição observada pouco se assemelha ao modelo de Retamal (2008), conforme mostrado pelo Quadro 8, e os prazos para cada fase do modelo FDD estabelecidos por Prikladnicki, Willi e Milani (2014) não foram confirmados pelo objeto de estudo.

Quadro 8 – Práticas Observadas do Modelo FDD.

FASE DO FDD	PROCESSOS DO FDD	DESRIÇÃO DO PROCCESSOS	OBSERVAÇÃO DO OBJETO DE ESTUDO
Iniciação	Desenvolver um modelo abrangente	Realizar um estudo dirigido sobre o escopo do sistema e seu contexto.	Não é feito.
	Construir a lista de funcionalidades	Identificar todas as funcionalidades que satisfaçam os requisitos de maneira categorizada.	Não é realizada por completo, tampouco é categorizada.
	Planejar por funcionalidade	Planejar a ordem na qual as funcionalidades serão implementadas, baseada nas dependências entre elas, na carga de trabalho da equipe de desenvolvimento e também na complexidade das funcionalidades a serem implementadas.	Não segue um padrão de planejamento.
Construção	Detalhar por funcionalidade	Produzir os pacotes de trabalho através da produção dos diagramas de sequência para as funcionalidades, escrevendo-se os prefácios das classes e métodos.	Não é feito.
	Construir por funcionalidade	Produzir uma função com valor para o cliente, implementando os pacotes de trabalho, de forma que o código desenvolvido passe pelo teste de unidade e pela inspeção, para depois ser promovido à versão atual.	A entrega dos incrementos do produto não é frequente, sendo entregue de maneira funcional somente ao final do processo.

Fonte: o autor.

Por fim, a observação em relação ao modelo FDD mostra que a instituição não segue os preceitos da metodologia ágil, pois o fato de não entregar incrementos de valor gradualmente, já o descredencia de ser qualificado como ágil.

5. CONSIDERAÇÕES FINAIS

Durante o estudo, realizado no Centro de Processamento de Dados de uma IES pública brasileira, em seu principal *campus*, foi possível observar como estas instituições, com a cultura e a burocracia que as caracterizam, realizam seus processos de desenvolvimento de *softwares*, bem como perceber a forma como são gerenciados esses tipos de projeto.

Mesmo com a utilização de metodologias ágeis, que segundo Carvalho e Mello (2012) diminuem a possibilidade de insucesso dos projetos, o estudo mostrou que o cenário observado (Instituição do Governo de Administração Direta), em geral, atinge o sucesso parcial ou até mesmo o fracasso na entrega de seus projetos de *software*, pois conclui seus serviços com atrasos, apesar da satisfação dos clientes, e alguns deles são até descontinuados. A Figura 2 e a Figura 3, mostram exatamente o que o estudo observou, que em organizações desse tipo a maior incidência dos projetos é de sucessos parciais (40,7%), seguida pelos fracassos (19,3%), e o atraso é um importante responsável por este diagnóstico.

Os motivos para estes projetos não terem conseguido entregar um produto/serviço com sucesso total (maior qualidade, dentro do prazo, escopo e orçamento atendidos, usuário satisfeito e valor agregado) decorrem da não completa aderência aos modelos de desenvolvimento de *software* (cascata e espiral); da fraca gestão de projetos, pois não há coleta de dados de métricas de projeto, nem há gestão de risco; e da adesão parcial das metodologias ágeis, pois prioriza o escopo ao tempo, isto é, as *sprints* não têm tempo fixo de execução, não utiliza ferramenta remota de comunicação com o cliente, e não entrega os incrementos do produto de maneira utilizável com frequência. Portanto, faz-se tudo de maneira incompleta na instituição do caso observado.

No entanto, esses pontos podem ser justificados pela discrepância entre as doutrinas aplicadas, pois as filosofias por trás dos modelos cascata, espiral e ágil são fundamentalmente diferentes. A aderência ou não a todos os modelos seria impossível. As metodologias ágeis utilizadas hoje na área do desenvolvimento de *software* estão atreladas a modelos mais avançados e modernos do que os tradicionais cascata e espiral. Portanto, respondendo à questão problemática do trabalho, não é possível dizer que o gerenciamento de projetos de *software* utilizado pela unidade estudada (CPD) esteja apto a desenvolver projetos padronizados, eficientes, com resultados efetivos e de sucesso a partir do emprego de metodologias ágeis. Tampouco, que metodologias ágeis conferem maior produtividade, ou que, o fato de ser elaborado numa instituição pública, trouxe implicações para o processo de desenvolvimento de *software*.

Como contribuições deste estudo podemos destacar:

- Mapeamento do processo de desenvolvimento de *software* (*as is*), que pode ser utilizado como parâmetro para os gestores que estejam buscando melhorar o processo;
- Entendimento da metodologia de desenvolvimento de *softwares*; e
- Identificação das deficiências que impedem o sucesso total dos projetos de *software*.

Sugere-se que em estudos posteriores o trabalho seja ampliado, propondo melhorias ao processo de desenvolvimento de *software*, ou seja, modelando um processo como ele deveria ser (*to be*). Pode-se também propor indicadores para auxiliar na gestão e no controle do processo.

REFERÊNCIAS BIBLIOGRÁFICAS

ARCHIBALD, R. D.; PRADO, D. **Pesquisa Maturidade em Gerenciamento de Projetos 2012 – Relatório Desenvolvimento de Novos Aplicativos (*Software*)**. Disponível em: <http://www.maturityresearch.com/novosite/2012/download/PesquisaMaturidade-2012_Relatorio-DNA-V2.pdf>. Acesso em: 09 abr. 2016.

ASSOCIAÇÃO BRASILEIRA DAS EMPRESAS DE SOFTWARE – ABES; *INTERNATIONAL DATA CORPORATION* – IDC. **Mercado Brasileiro de *Software* – Panorama e Tendências 2015**. Disponível em: <http://central.abessoftware.com.br/Content/UploadedFiles/Arquivos/Dados%202011/ABES-Publicacao-Mercado-2015-digital.pdf>. Acesso em: 30 mar. 2016.

AUDY, J. ***Scrum* 360**: um guia completo e prático de agilidade. 1. ed. Porto Alegre: Casa do Código, 2015.

CARVALHO, B. V.; MELLO, C. H. P. **Aplicação do Método Ágil S*crum* no Desenvolvimento de Produtos de Software em uma Pequena Empresa de Base Tecnológica**. Gestão & Produção. São Carlos, v. 19, n. 3, p. 557-573, 2012.

_____. **Revisão, Análise e Classificação da Literatura sobre o Método de Desenvolvimento de Produtos Ágil *Scrum***. XII Simpósio de Administração da Produção, Logística e Operações Internacionais – SIMPOI. São Paulo, 2009.

CENTRO DE INFORMÁTICA – CPD. **Estrutura Organizacional**. Disponível em: <http://www.cpd.unb.br/estrutura-organizacional>. Acesso em: 28 nov. 2016.

CUNNINGHAM. W.; et al. 2001. **Manifesto para Desenvolvimento Ágil de Software**. Disponível em: <http://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em: 28 nov. 2016.

FOINA, P. R. **Tecnologia de Informação**: Planejamento e gestão. São Paulo: Atlas, 2001.

FONSECA, F. E. A.; ROZENFELD, H. **Medição de Desempenho para a Gestão do Ciclo de Vida de Produtos**: Uma revisão sistemática da literatura. Revista Produção Online. Florianópolis, v.12, n. 1, p. 159-184, jan./mar. 2012.

GERHARDT, T. E.; SILVEIRA, D. T. **Métodos de Pesquisa**. 1. ed. Porto Alegre: UFRGS, 2009.

GIL, A. C. **Como Elaborar Projetos de Pesquisa**. 4. ed. São Paulo: Atlas, 2002.

GOLDENBERG, M. **A Arte de Pesquisar**: Como fazer pesquisa qualitativa em Ciências Sociais.11. ed. Rio de Janeiro: Record, 2009.

GOVERNO ELETRÔNICO. ***Software* Livre**. Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/*software*-livre>. Acesso em: 31 mar. 2016.

JUCÁ JUNIOR, A. S.; CONFORTO, E. C.; AMARAL, D. C. **Maturidade em Gestão de Projetos em Pequenas Empresas Desenvolvedoras de *Software* do Polo de Alta Tecnologia de São Carlos**. Gestão & Produção. São Carlos, v. 17, n. 1, p. 181-194, 2010.

KOTHE, P. B.; MARX, A. M.; FRANK, A. G. **Proposição de um Novo Método de Gerenciamento de Requisitos na Análise de Negócios**. Revista Produção Online. Florianópolis, v.15, n. 4, p. 1481-1510, out./dez. 2015.

LUDWIG, I. P.; ANZANELLO, M. J.; VIDOR, G. **Minimização dos Tempos de Atraso na Programação de Tarefas em uma Empresa de Desenvolvimento de *Softwares***. Revista Produção Online. Florianópolis, v.13, n. 2, p. 479-499, abr./jun. 2013.

MELO, C. O.; *et al*. ***The Evolution of Agile Software Development in Brazil****: Education, research, and the state-of-the-practice*. Journal of the Brazilian Computer Society – JBCS, v. 19, n. 4, p. 523-552, nov. 2013.

MIGUEL, P. A. C. **Estudo de Caso na Engenharia de Produção**: Estruturação e recomendações para sua condução. Produção, v. 17, n. 1, p.216-229, jan./abr. 2007.

OBJECT MANAGEMENT GROUP – OMG. ***Business Process Model and Notation* (BPMN)**. Versão 2.0. 2011a. Disponível em: <http://www.omg.org/spec/BPMN/2.0/PDF>. Acesso em: 29 abr. 2016.

_____. **OMG *Unified Modeling Language* (OMG UML) *Superstructure***. Versão 2.4.1. 2011b. Disponível em: <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/>. Acesso em: 29 nov. 2016.

PINTO, E. B.; VASCONCELOS, A. M.; LEZANA, A. G. R. **Abordagens do PMBOK e CMMI sobre o Sucesso dos Projetos de *Softwares***. Revista de Gestão e Projetos, v. 5, n. 1, p. 55-70, jan./abr. 2014.

PRESSMAN, R. S. **Engenharia de *Software***. 6. ed. São Paulo: McGraw-Hill, 2006.

PRIKLADNICKI, R.; WILLI, R.; MILANI, F. (Org.). **Métodos Ágeis para Desenvolvimento de Software**. Porto Alegre: Bookman, 2014.

RETAMAL, A. M. ***Feature-Driven Development***: descrição dos processos. Versão 1.2. Maio, 2008. Disponível em: <http://www.heptagon.com.br/files/FDD-Processos.pdf> Acesso em: 28 nov. 2016.

RUSCH, M.; OLIVEIRA, L. R. **Instrumento para Avaliação de Projetos na Gestão de Portfólio de Empresas Desenvolvedoras de *Software***. Revista Gestão & Tecnologia. Pedro Leopoldo, v. 12, n. 2, p. 113-140, jul./nov. 2012.

SCHWABER. K.; SUTHERLAND. J. **Guia do *Scrum***: as regras do jogo. Jul. 2013. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 28 nov. 2016.

SELLTIZ, C.; et al. **Métodos de Pesquisa nas Relações Sociais**. São Paulo: Herder, 1965. Reimpressão.

SERPRO. **A Empresa**. Disponível em: <http://www.serpro.gov.br/sobre/a-empresa>. Acesso em: 04 jul. 2016.

_____. **Demonstração dos Fluxos de Caixa 2015**. Disponível em: <http://www.serpro.gov.br/sobre/transparencia/contas-anuais/2015/demonstracao-dos-fluxos-de-caixa-2015/view>. Acesso em: 04 jul. 2016.

ANEXO A – Indicadores para Tecnologia da Informação

Quadro 9 – Indicadores para Tecnologia da Informação.

PERSPECTIVA	OBJETIVOS	INDICADORES
Contribuição para o negócio	Alinhamento ao negócio/TI	- Plano operacional/aprovação de orçamento.
	Valor entregue	- Desempenho das unidades.
	Gerenciamento de custo	- Realização das metas de despesa e entradas.
Orientação para clientes	Satisfação do usuário	- Nível de satisfação; - Qualidade do serviço; - Qualidade de atendimento e resposta; - Valor das recomendações e suporte de TI; - Contribuição para os objetivos de negócio.
	Custos competitivos	- Realização de custos unitários por profissional; - Taxa de trabalho em comparação a fornecedores externos.
	Desempenho dos serviços de desenvolvimento	- Taxa de sucesso do projeto; - Realização dos objetivos/metas propostos, incluindo custos, prazos, qualidade, escopo e gerenciamento; - Índice de satisfação dos patrocinadores de projeto; - Nível de gerenciamento do projeto.
	Desempenho dos serviços operacionais	- Realização dos níveis de serviço estabelecidos.
Orientação futura	Gerenciamento de recursos humanos	- Quadro de pessoal por habilidade; - Rotação de pessoal; - Taxa de pagamento de pessoal; - Dias de treinamento por membro da equipe.
	Satisfação dos funcionários	- Compensação; - Clima de trabalho; - Feedback; - Crescimento pessoal.
	Gestão do conhecimento	- Pesquisa em tecnologias emergentes; - Lições aprendidas.
Experiência operacional	Desempenho do processo de desenvolvimento	- Produtividade; - Qualidade; - Taxa de entrega.
	Desempenho do processo operacional	- Produtividade; - Capacidade de resposta; - Gestão de mudança.

continua

continuação

PERSPECTIVA	OBJETIVOS	INDICADORES
Experiência operacional	Experiência com o processo	- Planejamento; - Aquisição e implementação; - Entrega e suporte; - Monitoramento.
	Gerenciamento da arquitetura	- Aquisição.

Fonte: Grembergen e Haes (2009, apud RUSCH; OLIVEIRA, 2012).

ANEXO B – Valores e Princípios do Manifesto Ágil

Quadro 10 – Valores do Manifesto Ágil.

MAIS VALORIZADOS	MENOS VALORIZADOS
Indivíduos e interações	Processos e ferramentas
Software em funcionamento	Documentação abrangente
Colaboração com o cliente	Negociação de contratos
Responder a mudanças	Seguir um plano

Fonte: Cunningham (2001).

Quadro 11 – Princípios do Manifesto Ágil.

	PRINCÍPIOS
1	Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.
2	Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.
3	Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.
4	Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.
5	Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.
6	O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.
7	Software funcionando é a medida primária de progresso.
8	Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.
9	Contínua atenção à excelência técnica e bom design aumenta a agilidade.
10	Simplicidade (a arte de maximizar a quantidade de trabalho não realizado) é essencial.
11	As melhores arquiteturas, requisitos e designs emergem de equipes auto organizáveis.
12	Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Fonte: Cunningham (2001).

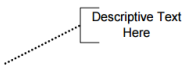
ANEXO C – Elementos do BPMN

Quadro 12 – Elementos do BPMN.

CATEGORIA	ELEMENTO	DESCRIÇÃO	NOTAÇÃO
Objetos de Fluxo	**Evento**	Um evento é algo que "acontece" durante o andamento de um processo. Estes eventos afetam o fluxo do modelo e, geralmente, têm uma causa (lançamento) ou um impacto (resultado). Os eventos são representados por círculos com centros abertos para permitir internamente indicar os diferentes lançamentos ou resultados. Existem três tipos de eventos: Inicial, Intermediário ou Final.	
	Atividade	Uma atividade é um termo genérico para o trabalho que a empresa realiza em um processo. As atividades são representadas por retângulos arredondados, e podem ser do tipo tarefa ou subprocesso.	
	Decisão	A decisão é usada para controlar a divergência e convergência de fluxos de sequência de fluxos de um processo. Assim, será determinada a ramificação, bifurcação, fusão e união dos caminhos. A decisão é representada por um losango e internamente é indicado o tipo de controle de comportamento.	
Conexões	**Fluxo de sequência**	O fluxo de sequência é usado para mostrar a ordem na qual as atividades são realizadas no processo. De tal maneira que é representado por uma seta contínua, fechada e cheia.	
	Fluxo de mensagem	O fluxo de mensagem é usado para mostrar a comunicação entre dois participantes que estão preparados para enviar e receber mensagens. De modo que é representado por uma seta tracejada, fechada e vazia.	
	Associação	Uma associação é usada para ligar informações e artefatos com qualquer elemento gráfico do BPMN. Anotações de texto e outros artefatos podem ser associados com elementos gráficos. Dessa tal forma que é representada por uma seta pontilhada e aberta.	
Dados	**Objeto de dados**	Os objetos de dados fornecem informações sobre quais atividades precisam ser realizadas e/ou o que elas produzem. Os objetos de dados podem ser representados por um objeto singular ou por uma coleção de objetos.	

continua

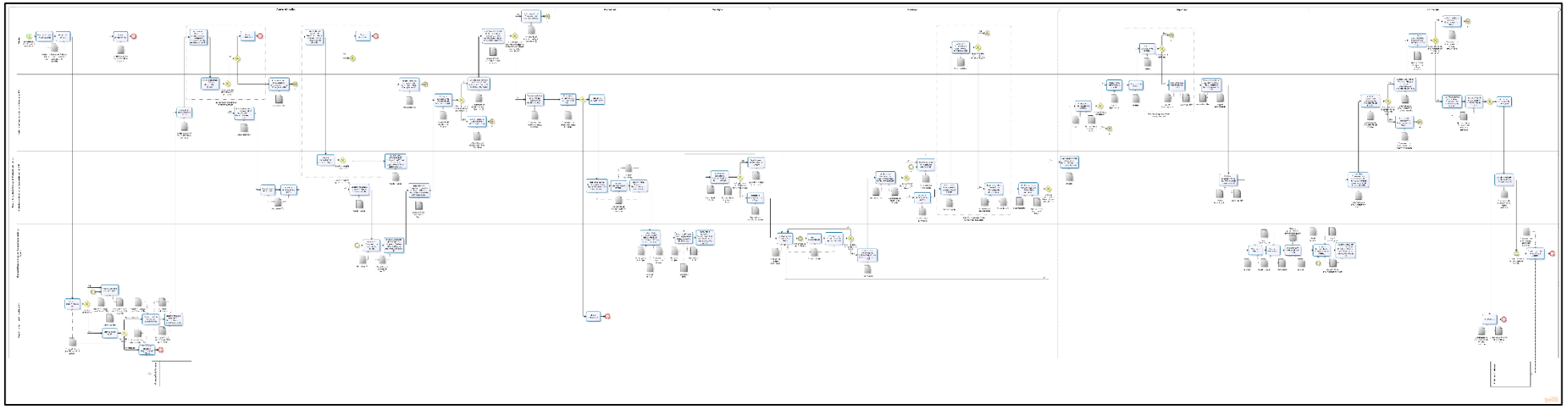
continuação

CATEGORIA	ELEMENTO	DESCRIÇÃO	NOTAÇÃO
Divisões	**Piscina**	A piscina é a representação gráfica dos participantes de um processo. Ela atua, também, como uma raia de um conjunto de atividades de outras piscinas. A piscina pode conter detalhes internos, sob a forma do processo que irá ser executado. Ou não possuir detalhes internos, isto é, será uma "caixa preta".	
	Raia	Uma raia é uma subdivisão, no interior de uma piscina, dentro de um processo. As raias são utilizadas para organizar e categorizar as atividades.	
Mensagem	**Mensagem**	A mensagem é usada para representar o conteúdo de uma comunicação entre dois participantes.	
Artefatos	**Grupo**	Um grupo é um agrupamento de elementos gráficos que estão dentro da mesma categoria. Este tipo de agrupamento não afeta o fluxo de sequência. O nome da categoria aparece no diagrama como o título do grupo. Os grupos são uma maneira de como as categorias de objetos podem ser exibidas visualmente no diagrama. Assim, os grupos são representados por uma caixa pontilhada em torno de um bando de objetos dentro da mesma categoria).	
	Anotação de texto	As anotações de texto são um mecanismo que permite ao modelador fornecer informações de texto adicionais para o leitor de um diagrama BPMN de modo que são ligadas por uma associação.	

Fonte: OMG, 2011a.

ANEXO D – Processo de Desenvolvimento de Sistemas em um Órgão Público

Figura 23 – Processo de Desenvolvimento de Sistemas em um Órgão Público.



Fonte: o autor.

ANEXO E – Documento de Abertura do Projeto

Figura 24 – Documento de Abertura do Projeto.

<logo Instituição>	<logo Centro de Informática>	<Sigla do Projeto - Nome do Projeto> **Documento de Abertura do Projeto** **Solicitação de Sistemas**	<cod. projeto>	<logo Cliente>

1. Descrição deste documento

Este documento pretende registrar as informações básicas do projeto, dados iniciais necessários à avaliação da viabilidade do projeto, bem como o princípio do levantamento de requisitos. Desta forma, é possível delimitar objetivos, benefícios alcançados, justificativa de implementação, fronteiras, integrações etc.

2. Solicitante

[Deve conter a identificação do solicitante do sistema, contendo: Nome, Unidade/Sigla, vínculo com a UnB, matrícula, E-mail, telefone fixo, telefone celular.]

3. Nome sugerido para o Sistema

[Coloque aqui o nome pelo qual o Sistema deve ser conhecido.]

4. Objetivo(s)

[Relacione o objetivo geral e os específicos do projeto, ou seja, a razão de ser e para quê, o que se pretende com a execução do projeto. Um projeto só é considerado bem sucedido quando todos os seus objetivos são atingidos.]

5. Escopo e Não escopo

5.1. Escopo

[O escopo expressa a "extensão" ou "amplitude" do projeto (em termos do que se pretende realizar, abarcar ou abranger). Estabelece o seu "raio de ação" ou "cobertura", definindo, portanto, seus "limites". O escopo é, em síntese, a alma do projeto, porque expressa sua essência e identidade. Desta forma, devem ser registrados todos os aspectos e funcionalidades que devem fazer parte do projeto e que de alguma forma irão afetar os procedimentos das áreas envolvidas direta ou indiretamente. É importante que neste tópico fique bem explícito o universo e o limite de fronteiras do projeto.]

[É importante ressaltar que no escopo deve(m) estar especificado(s), dentro dos processos de negócio mapeados, aquele(es) que, prioritariamente, deverá(ao) ser automatizados.]

5.2. Não-escopo

[Descrever todos os aspectos e funcionalidades que não fazem parte do projeto, que estão excluídos do escopo. Esta informação é importante para o posterior gerenciamento das expectativas do cliente.]

6. Justificativa

[Deve conter a fundamentação, ou seja, a motivação pela qual há a necessidade da existência do projeto. Com base nos objetivos, explique o porquê e a situação atual do processo de negócio executado para cada um destes objetivos, se necessário. Esta seção deve fazer referência também aos motivos que conduziram à escolha da automação dos processos referenciados na seção "Escopo".]

7. Benefícios a serem alcançados

[Descreva de forma clara quais os benefícios, vantagens e expectativas a serem atingidos. Estão diretamente relacionados aos objetivos específicos.]

8. Documentos Anexados

[Nesta seção deve ser anexada a documentação resultante do mapeamento dos processos de negócio, os documentos referentes à legislação pertinente e outros que sejam necessários para a melhor compreensão da demanda.]

[ATENÇÃO: Os textos em azul são para auxiliar o preenchimento do artefato, devendo ser excluídos ao final da elaboração do registro.]

Data da Solicitação	Assinatura e carimbo do responsável pela Unidade

Fonte: o objeto de estudo.