CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

PRINCIPAIS RAZÕES DOS PROJETOS DE TI RESULTAREM EM FRACASSO

WANDERSON GUIMARÃES SOFFE

Americana, SP
2013

CENTRO PAULA SOUZA
GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Americana
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

PRINCIPAIS RAZÕES DOS PROJETOS DE TI RESULTAREM EM FRACASSO

WANDERSON GUIMARÃES SOFFE
wsoffe@gmail.com

Trabalho Monográfico, desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Fatec-Americana, sob orientação do Prof. Alberto Martins Júnior.

Área: Análise e Desenvolvimento de Sistemas

Americana, SP
2013

BANCA EXAMINADORA

Prof. Me. Alberto Martins Júnior
(Orientador)

Prof. Esp. Nivaldo Tadeu Marcusso
(Convidado)

Prof. Dr. Alexandre Mello Ferreira
(Convidado)

AGRADECIMENTOS

Primeiramente a Deus que esteve comigo em todos os momentos. Sem a sua ajuda não conseguiria chegar até aqui.

À minha esposa Milena, que me apoiou de forma decisiva na revisão desse projeto.

Ao meu orientador Alberto Martins Júnior por confiar em minha proposta e ter aceitado de bom grato ser meu orientador.

DEDICATÓRIA

A Deus, aos meus pais, pelo apoio e paciência, aos meus amigos e familiares que estão no Rio de Janeiro e que eternamente estarão em minhas lembranças.

RESUMO

Ao contrário de outros ramos profissionais, onde os produtos entregues ao término de um projeto são tangíveis, o software é um produto abstrato e intangível, portanto muito mais susceptível a falhas. Este trabalho visa elicitar as principais causas de fracasso de um projeto de TI. Esses problemas são originários principalmente de fatores como custo, tempo e aceitação do cliente (escopo e qualidade). Mediante a esses problemas será proposta possíveis soluções em cada uma dessas áreas para minimizar esses riscos. Essas soluções são oriundas de metodologias de gerência de projeto, engenharia de software e programação orientada a objetos. Essas metodologias podem prevenir a maioria dos projetos de software de eventuais fracassos.

Palavras Chave: Gerenciamento de Projetos; Engenharia de Software; e Programação Orientada a Objetos

ABSTRACT

Unlike other professional fields, where the products delivered at the end of a project are tangible, the software product is an abstract and intangible, so much more susceptible to failure. This paper aims to elicit the main causes of failure of a software project. These problems originate mainly from factors such as cost, time and customer acceptance. Through these problems will be proposed possible solutions in each of these areas to minimize these risks. These solutions are derived from methodologies of project management, software engineering and object-oriented programming. These methodologies can prevent most software projects of any failures.

Keywords: Project management; software engineering; object-oriented programming.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 - Introdução

Em um mundo cada vez mais globalizado e competitivo, é cada vez mais difícil desenvolver sistemas de informação de alta complexidade. Os prazos de entrega estão mais curtos, o orçamento inicial subestimado, a qualidade está mais questionável, mão de obra heterogenia e outros fatores externos podem trazer consequências desastrosas para o sucesso de um projeto de TI, podendo em muitos casos resultar em fracasso.

Para Yourdon (1996) a tendência de um projeto de TI[1] terminar em fracasso não é um caso isolado, ao contrário, é uma regra e não exceção. Porém mesmo sabendo desse problema, muitos desenvolvedores, gerentes e outros profissionais se envolvem nesses projetos.

Mas por que esses profissionais se dedicam a esses projetos? Existem muitas variáveis para responder a essa questão que serão explicadas ao longo desse trabalho. As pressões internas e externas ao projeto são sem dúvida alguma uma das principais causas de envolvimento de profissionais em tais projetos virtualmente impossíveis.

Não existe uma metodologia totalmente eficaz de tornar um projeto com alto risco de fracasso se reverter em sucesso. Se existisse essa metodologia, provavelmente não haveria tantos relatos de projetos de software milionários que nunca foram concluídos com sucesso.

É notório que as principais causas de fracasso de um projeto de TI estão diretamente relacionadas às seguintes restrições: **Custo, Prazo, Qualidade e Escopo.** Essas restrições serão estudadas em detalhes nesse trabalho.

O que pode minimizar esses riscos são métricas de boas práticas de programação associadas à gerência de projetos responsável, engenharia de

[1] Tecnologia da Informação

software eficaz e reaproveitamento de código através do paradigma de POO[2]. É possível que mesmo se utilizarmos essas métricas um projeto possa resultar em fracasso, porém esses riscos tendem a cair consideravelmente.

1.1- Justificativa

Em outras atividades profissionais, quando o cliente solicita um produto ou serviço, ele geralmente terá a garantia de que receberá o produto exatamente como quer, pagando pelo orçamento previamente acordado e será entregue no prazo estipulado pelo prestador de serviço ou fornecedor.

Por exemplo, quando um cliente pedir um serviço de pintura residencial, ele terá a garantia que a cor pedida será branca e não terá a surpresa de encontrá-la na cor rosa. Ele terá a garantia que o orçamento acordado será de R$1.500,00 e não R$15.000.00. E por fim o cliente terá a garantia que a pintura será entregue no prazo máximo de uma semana e não em seis meses. (VIEIRA, 2003)

O tema desse trabalho se justifica porque ao contrário dos exemplos citados acima, muitos projetos de software são concluídos com orçamentos muito acima do que previamente acordado, prazos de entrega não cumpridos e principalmente o produto (software) entregue não atende às reais necessidades dos clientes. (YOURDON 1996).

Esses problemas são resultados principalmente da complexidade de desenvolver um software de grande porte. Soma-se a isso a documentação do projeto insuficiente, tecnologias inadequadas, recursos humanos não qualificados, prazos subestimados e orçamentos insuficientes para continuidade do projeto.

Segundo Vargas (2009), outros problemas podem resultar em fracasso em projeto de TI em nível de gerência de projeto: Falta de liderança do gerente de projeto; Falta de conhecimento de pontos chaves do projeto; Falta de compreensão

[2] Programação a Orientada a Objetos

em relação às necessidades do cliente; Múltiplas lideranças em posições conflitantes.

A partir das questões mencionadas acima surge então a seguinte pergunta: Como evitar que um projeto de TI resulte em fracasso?

1.2- Objetivo Geral e Objetivo Especifico

Este trabalho tem como **objetivo geral:** elicitar as principais causas de um projeto de TI que resultam em fracasso e propor soluções nas áreas de engenharia de software, gerenciamento de projetos e UML[3].

Muitas dessas soluções propostas já são conhecidas no meio profissional e até mesmo acadêmico, porém a falta de conhecimento dessas métricas é muito grande.

Os **objetivos específicos** vão realizar um levantamento bibliográfico sobre as principais causas de um projeto de TI resultar em fracasso e também as principais metodologias de boa prática de programação, engenharia de software e gerência de projeto.

1.3- Hipóteses

Como hipóteses, podemos mencionar as boas práticas de programação utilizando a programação orientada a objetos, com o objetivo de reuso de código através de componentes de software, *Frameworks*, *Designers Patterns* (SOMMERVILLE, 2008), e gerência de projetos (VARGAS 2009).

Outra hipótese que podemos levar em consideração, como por exemplo, é o gerenciamento de projetos baseado na metodologia do PMBOK. As hipóteses acima mencionadas podem resultar em afirmativa, provável e negativa.

[3] *Unified Modeling Language*

1.4- Metodologia

Como metodologia para o desenvolvimento desse trabalho, foi utilizada a pesquisa bibliográfica em livros tecnológicos que enfocam os temas propostos, consulta de artigos e acesso a endereços eletrônicos com referências reconhecidamente idôneos, onde o pesquisador fundamenta suas teses em fontes do meio acadêmico.

Em todos os capítulos serão detalhados os problemas mais comuns e suas consequências mais visíveis para o fracasso de um projeto de TI, juntamente com suas possíveis soluções.

O trabalho foi estruturado em **sete capítulos**, sendo que o **terceiro** conceitua os problemas referentes ao orçamento de um projeto, o **quarto** menciona os problemas referentes ao prazo de entrega do produto final, o **quinto** descreve os problemas de qualidade, ou seja, o conflito entre o produto entregue e as expectativas do cliente, o **sexto** aborda os problemas do escopo do projeto de TI.

Com base nas informações conseguidas a partir dos estudos realizados nos capítulos anteriores, o **sétimo** capítulo se reserva às **Considerações Finais**.

2- Revisão da Literatura

Neste capítulo serão apresentados os conceitos principais de um projeto, da disciplina de gerenciamento de projetos, software, engenharia de software e a definição de um fracasso.

2.1- Projeto

Vargas (2010) afirma que um projeto pode ser definido como “Um empreendimento não repetitivo caracterizado por uma sequência lógica e clara de eventos com início, meio e fim, que se destina a fazer a atingir um objetivo claro e definido, sendo conduzido por pessoas dentro de parâmetros predefinidos de tempo, custo, recursos, envolvimento e qualidade”.

Um projeto pode ter várias aplicações. Entre elas podemos destacar: construções de pontes, estádios de futebol, organização de festa de casamento, festa de formatura e debutantes, etc.

2.2 – Gerenciamento de Projetos

Segundo Dinsmore, Brewin (2009), o gerenciamento de projetos pode ser definido como o ramo de conhecimento que auxilia no monitoramento, execução, e controle de um projeto.

Dinsmore, Brewin (2009) acrescentam o gerenciamento de projeto se preocupa com as alocações de recursos, monitoramento da maturidade dos processos, fatores de riscos que podem comprometer o sucesso do projeto, economia de recursos sem prejudicar o andamento dos processos, a garantia de que o cliente receba exatamente o que pediu (qualidade e escopo), a integração de todos os processos em uma única solução.

2.2.1 – Áreas do Gerenciamento de Projeto

Seguindo Vargas (2010), o PMBOK[4] *Guide*, atualmente em sua quarta edição é um guia desenvolvido pelo PMI[5], onde abrange as nove grandes áreas de conhecimentos que auxiliam no gerenciamento do projeto. São elas:

- Gerenciamento de Escopo
- Gerenciamento de Tempo
- Gerenciamento de Custo
- Gerenciamento de Qualidade
- Gerenciamento de Integração
- Gerenciamento de Recursos Humanos
- Gerenciamento de Comunicação
- Gerenciamento de Riscos
- Gerenciamento de Aquisições

Siegelaub (2011) acrescenta que das nove áreas citadas acima, três causam impacto direto no projeto – Gerenciamento de Custo, Gerenciamento de Tempo e Gerenciamento de Escopo, onde são chamadas de tríplice restrição do projeto. A tríplice restrição será estuda nos próximos capítulos somados ao gerenciamento de qualidade.

2.3 – Software

De acordo com Sommerviller (2007), ao contrário do que muitos pensam, o software não é apenas um programa de computador, mas também todos os processos, documentações e configurações que auxiliam o funcionamento de um programa.

Como exemplo, podemos citar o banco de dados e sua respectiva base de dados. Em relação às documentações - os manuais do usuário auxiliam na

[4] *Project Management Body of Knowledge*
[5] *Project Management Institute*

compreensão do funcionamento do programa. E por fim as configurações para o melhor funcionamento do sistema segundo as necessidades do usuário.

2.4 – Tecnologia da Informação

Alecrim (2011) explica que a Tecnologia da Informação é definida como um conjunto de dados que necessitam ser processadas por recursos computacionais, ou seja, um conjunto de dados que são processados e convertidos em informação.

2.5 – Engenharia de Software

Sommerviller (2007) ensina que a Engenharia de Software é uma disciplina de engenharia que auxilia no desenvolvimento de softwares complexos. Desde a coleta de requisitos, planejamento, implementação, implantação e manutenção.

2.6 – Definições de Fracasso

Esse trabalho considera o termo “fracasso” baseando em três dicionários eletrônicos reconhecidos como idôneos: Houaiss, Aulete e Michaelis.

De acordo com a definição do dicionário Houaiss (web), o fracasso pode ser definido como ausência de êxito, derrota. Como por exemplo, “Esforçou-se muito no projeto, porém foi um fracasso”.

Para o dicionário Aulete (web), o fracasso é a falta de êxito na profissão ou em qualquer outra dimensão da vida.

Por fim, o dicionário Michaelis (web) menciona que o fracasso é definido como uma ruína, desgraça, insucesso ou um mau êxito.

3. Custos e Orçamento de um Projeto

Os custos e orçamentos de um projeto de TI é um fator determinante para o sucesso ou fracasso. De acordo com o artigo publicado (LEMOS VIEIRA, pág. 25, 2003), o aumento de preço de um projeto ainda em execução é de até 45%. São estatísticas preocupantes, esse aumento representa o prejuízo que será repassado diretamente ao cliente gerando insatisfação.

3.1- Orçamentos Subestimados

Segundo Yourdon (1996), um dos grandes motivos de fracasso de um projeto de TI são orçamentos muitas vezes subestimados. Isso decorre, principalmente, por contenção de despesas, concorrência feroz em um mercado altamente competitivo.

Como consequência desse corte, o quadro de colaboradores é cortado sendo insuficiente para cumprir o cronograma. Outra consequência para os orçamentos subestimados é a sobrecarga de trabalho para equipe, aumentando consideravelmente a pressão para conclusão do projeto.

Outras consequências que podemos citar é o prejuízo iminente se a empresa mantiver esses orçamentos ou na pior das hipóteses, refazer os orçamentos gerando quebra de contrato.

Como em qualquer outro segmento profissional o orçamento de um projeto de TI é calculado aplicando os custos do projeto somando-se ao lucro da empresa. Porém, Sommerviller (2007) acrescenta que cálculo do custo de um projeto de TI é um processo complexo.

Sommerviller (2007) complementa que muitas empresas iniciantes no mercado de TI cotam orçamentos abaixo do mercado com a intenção de entrar em um novo segmento, mesmo obtendo um lucro pequeno. Esta é uma prática de alto risco, porém muitas vezes necessária.

Outro problema apontado por Sommerviller (2007) são os orçamentos subestimados com o intuito de vencer concorrências ou licitações. Esse problema é muito comum devido a dificuldades financeiras pelas empresas.

3.2- Estimativa de Custos

Pressman (1995) afirma que algumas empresas procuram realizar levantamento das estimativas de custos do projeto durante o processo do desenvolvimento, tentando com isso obter maior precisão.

Porém, esta prática não é funcional porque os clientes não irão fechar um contrato enquanto não tiver um orçamento em mãos. E isso geralmente ocorre no início do processo.

Sommerviller (2007) acrescenta que a melhor forma de planejar os custos de um software é elaborar um conjunto de três métricas que ajudam realizar uma estimativa de custos de software. Estas métricas são os principais pilares para orçar um projeto de TI:

- Estimativa de Custo do Esforço Empregado por Atividade
- Tempo necessário para completar cada atividade.
- Custo total de cada atividade.

3.2.1 - Estimativa de Custo do Esforço Empregado por Atividade

Segundo Sommerviller (2007), o principal parâmetro de custo é o esforço empregado em cada atividade, porém o cálculo para se estimar esse custo não é uma tarefa fácil e com possíveis resultados imprecisos.

Pressman (1995) alerta que um grande erro na estimativa de custo de um software pode ser suficiente para determinar o sucesso ou fracasso. Para se calcular a estimativa de esforço empregado por atividade, primeiramente é preciso separar do escopo do projeto as funções. Em seguida fazer uma análise dos requisitos, codificação e testes de qualidade.

A partir de então o analista realiza o cálculo de esforço (pessoas/mês) onde será verificado por cada atividade o esforço empregado.

Outro fator determinante é a aptidão individual dos desenvolvedores envolvidos no projeto. Um desenvolvedor de nível Sênior certamente encontrará soluções de desenvolvimento em tempo menor em relação a um profissional de nível Júnior.

O tempo de projeto deve ser estimado juntamente com os custos gerais do sistema, sendo comumente chamado de prazo de projeto.

Segundo Sommerviller (2007), é importante salientar que o aumento do número de desenvolvedores no projeto não significa necessariamente que o projeto de TI terminará no prazo. Fatores de logística e até mesmo da estrutura física podem dificultar essa abordagem.

3.2.2 - Modelo de custo COCOMO

Para se obtiver estimativas de custos mais precisas vários modelos de algoritmos de custo estão disponíveis.

Entretanto o algoritmo COCOMO[6] é de acordo com Sommerviller (2007) que um modelo que leva em consideração vários atributos que podem influenciar no custo de um projeto de TI. Esses atributos incluem custo de projeto, hardware e pessoal. Uma das vantagens do modelo COCOMO é, a adaptabilidade de algoritmos de acordo com a complexidade e compreensão dos projetos.

3.2.2.1 - Modelo COCOMO 81

Segundo Sommerviller (2007), modelo COCOMO 81 é baseado no processo cascata de desenvolvimento podendo trabalhar em três níveis de complexidade.

[6] *Constructive Cost Model*

Porém o modelo COCOO 81 não prevê a utilização de artefatos em prototipação em suas fórmulas, tornando-se ultrapassado para projetos que utilizam essa tecnologia.

A tabela 1 se refere à versão COCOMO 81 para processos em cascata

Complexidade	Fórmula	Descrição
Simples	PM=2.4 (KDS) $^{1.05}$ x M	Aplicação simples, de boa compreensão desenvolvida por pequenas equipes.
Moderada	PM=3.0 (KDS) $^{1.12}$ x M	Projetos de média complexidade e com média compreensão dos envolvidos no projeto.
Incorporada	PM=3.6 (KDS) $^{1.2}$ x M	Projetos de alta complexidade com o software é fortemente acoplado a hardware, leis e outros procedimentos operacionais.

Tabela 4 – Fórmulas para processos em cascata modelo COCOMO81 – Fonte Sommerviller (2007)

Onde **PM** é o esforço estimado em pessoas mês. **M** refletem as características do produto projeto.

3.2.2.2 - Modelo COCOMO II

O modelo COCOMO II é baseado no processo espiral de desenvolvimento podendo reconhecer diversos submodelos:

- Modelo de composição de aplicações: é um modelo baseado em componentes reusáveis srcreans ou banco de dados. Sua fórmula de aplicação é **PM= (NAP x (1 - % reuso/100)) /PROD**. Onde **PM** é o esforço estimado em pessoas/mês; **NAP** é o número total de aplicações; **% reuso** é porcentagem de código reutilizado na aplicação e **PROD** é a produtividade em pontos do objeto.
- Modelo de projeto preliminar: indicado para projetos iniciais após os requisitos serem validados. Sua fórmula de aplicação é:

esforço=A x tamanho^{b} x M, onde o coeficiente **A** é uma constante com valor fixo de 2,94; **b** é o expoente que reflete o crescente nível de esforço e **M** é o multiplicador que se baseia em um conjunto de processos e projetos.

- Modelo de reuso: utilizado para calcular o esforço necessário para integrar os componentes reusáveis na aplicação.

3.2.3 - Outros custos no Projeto

Outros custos que não são diretamente ligados aos custos de desenvolvimento devem ser levados em consideração.

- Custos de hardware
- Custos de ferramentas de desenvolvimento
- Custo de viagens de reuniões com *Stakeholders*
- Custos de ferramentas de desenvolvimento
- Custos de pessoal de apoio
- Custos de aluguel das instalações da empresa

Alguns desses custos secundários podem ser facilmente incorporados em outros projetos. Como resultado, ajudam a diminuir os custos de um projeto de TI como, por exemplo, custos de hardware e ferramentas de desenvolvimento.

Vargas (2009) afirma que um plano de projeto, o gerenciamento de custos e o gerenciamento de aquisições são necessários para um melhor controle do capital disponível para obter os recursos necessários para um bom planejamento.

3.3 - Reuso para minimizar os custos

Segundo Sommerviller (2007), o reuso é sem dúvida uma das formas mais eficientes de diminuir os custos de software, aumentar os lucros e até mesmo diminuir o tempo de entrega do produto final.

Existem diversas formas de reuso que vão de acordo com a necessidade do projeto. Um projeto pode utilizar mais de uma solução de reuso como, por exemplo, *Frameworks,* e componentes.

3.3.1 - Reutilização de Código

A reutilização de código é uma abordagem útil e muito utilizada para redução de custos no desenvolvimento de um projeto de TI. Sommerviller (2007) acrescenta que a abordagem de reuso possui as seguintes abordagens: paradigma de programação orientada a objetos, paradigma de programação estruturada, componentes, *Frameworks*, *Designer Patterns e r*euso de sistemas. Todas essas abordagens serão analisadas nessa sessão.

3.3.1.1 - Paradigma de Programação Orientada a Objetos

Segundo Paula Filho (2009) a reutilização de código em programação orientada a objetos pode ser dividida em:

- Reutilização de Métodos / Funções;
- Reutilização objetos;

Deitel (2010) explica que um método ou função podem realizar uma tarefa podendo ou não retornar um resultado. Os métodos podem conter parâmetros opcionais que representam informações adicionais para o método.

O método está contido em uma classe que poderá conter outros métodos. Esses métodos podem ser chamados em outras classes através da instância de objetos.

É através dessa instância de objetos que obtemos a reutilização de código na POO. Esses objetos podem conter de forma opcional parâmetros externos.

Os conceitos de POO são vastos, contendo vários conceitos que contribuem para o reaproveitamento de código, DEITEL (2010).

- **Herança -** A herança é o conceito mais difundido de programação orientada a objetos, onde uma superclasse ou classe pai fornece todos os seus métodos para classes filhas. Este conceito contribui de forma bem eficiente e simples para o reaproveitamento de código. Algumas linguagens de programação como C++ permitem a herança múltipla, ou seja, uma classe filha pode herdar de duas superclasses.
- **Polimorfismo –** O polimorfismo permite que um mesmo método possa ser implementado de várias formas diferentes com retornos e parâmetros diferentes. O polimorfismo é útil para o reaproveitamento de código porque é possível implementar sistemas extensivos.
- **Abstração –** Outra forma de polimorfismo, a abstração, é utilizada em nível de hierarquia de classes. As classes abstratas não podem ser utilizadas para instanciar objetos, porém podemos criar métodos concretos e abstratos que não podem ser implementados na classe principal.
- **Interfaces -** As interfaces controlam quais as operações são permitidas e como são realizadas bem como, o conjunto de métodos que pode ser chamado pelo objeto.

Apesar da inovação em reuso de código através do paradigma orientado a objetos, sua implementação é mais complexa em relação ao paradigma de programação estruturada, encontrando alguma resistência por parte de desenvolvedores.

As principais linguagens de POO são C++, Java, C# e *Object* Pascal (Delphi), sendo que essa última possui comportamento híbrido, ou seja, contém características de programação orientada a objetos e programação estruturada.

3.3.1.2 - Paradigma de Programação Estruturada

É importante observar que além da programação orientada a objetos, a reutilização de códigos pode ser aplicada ao paradigma de programação estruturada através de funções.

A reutilização de código no paradigma de programação estruturada não é tão eficiente quanto à programação orientada a objeto, porém é muito popular e de fácil aplicação.

As principais linguagens de programação estruturada são Pascal, COBOL, *Fortran* e C, sendo que essa última é à base das principais linguagens de programação orientada a objetos.

3.3.2 - Componentes

Segundo Sommerviller (2007), um componente pode ser definido como uma unidade de software independente e fracamente acoplada. Graças a essas características os componentes podem ser reusados em outros processos de software ou em até mesmo em outros projetos diminuindo os custos do projeto.

Devido à alta complexidade em que os softwares são desenvolvidos, o uso dos componentes tornou-se uma ferramenta primordial para diminuição de custos.

A funcionalidade de um componente e suas dependências é definida a partir de um conjunto de interfaces públicas, podendo ser combinados com outros componentes, sendo implantados de forma única.

Os componentes são em sua maioria utilizada em sistemas de informação de grande complexidade, como por exemplo, o *E-commerce*. Porém algumas empresas de TI possuem objeções quanto ao seu uso devido a questões de segurança.

É importante observar que os componentes podem ocasionar problemas caso utilizados de forma incorreta. Podemos listar a seguir os mais significativos.

- **Confiabilidade –** muitos componentes não disponibilizam o seu código fonte, gerando desconfiança aos desenvolvedores.

- **Certificação -** outro fator importante à certificação de componentes apesar de ter uma avaliação independente para assegurar sua legitimidade, não transmite confiança necessária, pois não existem garantias necessárias.

Apesar desses problemas os componentes são sem dúvida uma boa solução para reutilização de código e consequentemente redução de custos de software.

3.3.3 - *Frameworks*

Sommerviller (2007) define os Frameworks como um conjunto de objetos concretos e abstratos projetados para serem utilizados na criação de outros objetos. Os *Frameworks* podem auxiliar na criação de subsistemas e aplicações mais específicas.

Sommerviller (2007) acredita que a programação orientada a objetos ficou inapropriado para aplicações de grande porte. Isso ocorre devido às suas abstrações pequenas e muito especializadas para uma aplicação especifica.

Os *Frameworks* surgiram para suprir essa carência, permitindo o reuso de grandes blocos de aplicações e facilitando a sua codificação. Podem ser classificados em três tipos:

- **Infraestrutura de sistema –** Utilizados basicamente para interface com o usuário e compiladores.
- **Aplicações empresariais –** Utilizados principalmente para aplicações específicas como comércio, finanças e negócios.
- **Aplicações de integração** – Conjunto de classes de objetos e padrões que permitem a troca de informações entre componentes. Como exemplo desses *Frameworks*, podemos citar o *Dot Net* da Microsoft, *Cake* Php, Java *Beans*, etc.

Sommerviller (2007) afirma que principal problema dos *Frameworks* é a sua dificuldade de compreensão devido a sua grande complexidade. Como

consequência, os *Frameworks* exigem profissionais especializados podendo com isso aumentar os custos em um primeiro momento.

3.3.4 – *Designer Patterns*

Sommerviller (2007) define os *Designer Patterns* como abstrações de alto nível para soluções em projetos. Essas soluções são baseadas em objetos como herança e polimorfismo que podem ser reusadas em outros projetos.

Existem muitos modelos de *Designer Patterns,* cuja utilização é de acordo com os requisitos de negócios de um projeto. Cada padrão pode ser representado graficamente pelo diagrama de classes em UML.

Abordaremos a seguir listar os *Designers Patterns* mais utilizados: criação por método, observador, fábrica e MVC[7].

3.3.4.1 – Criação por Método

Segundo Magela (2006), esse *Pattern* tem como objetivo a criação de objetos, deixando as subclasses decidirem qual objeto instanciar. Esse modelo permite que uma classe possibilite que a subclasse crie os objetos dinamicamente (Figura 1).

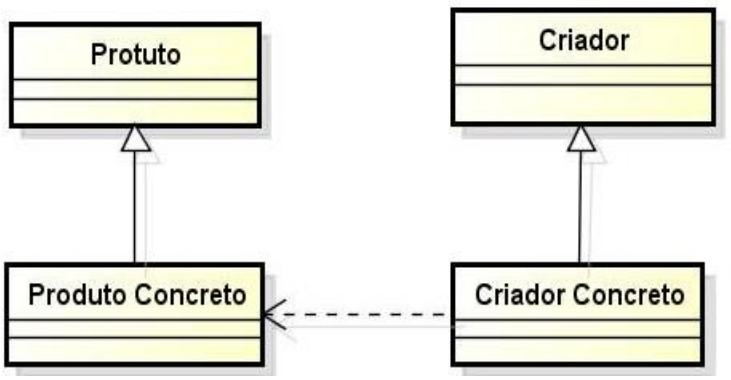

Figura 1 - Modelo Criação Método – Fonte http://www.oodesign.com

[7] *Model View Controller*

Esse *Pattern* é utilizado por *Frameworks* sendo geralmente aceito para aplicações bancárias. É útil também quando uma classe não pode saber com antecedência quais objetos deve-se instanciar.

3.3.4.2 – Padrão *Singleton*

Baseado no endereço http://www.oodesign.com (acessado em dez 2012), o padrão *Singleton* é muito popular, sendo utilizado quando há necessidade de usar apenas uma instância de classe (Figura 2).

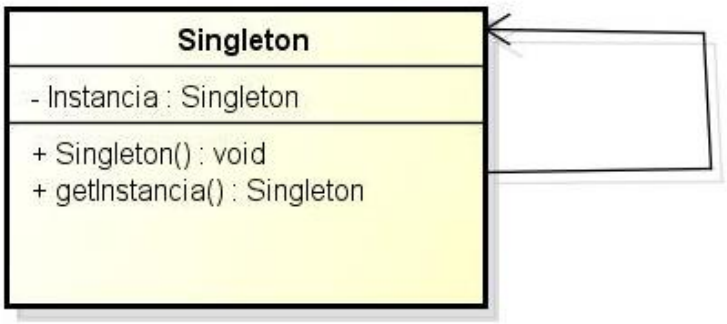

Figura 2 - Modelo *Singleton* – Fonte http://www.oodesign.com

3.3.4.3 – Padrão Fábrica

Baseado no endereço http://www.oodesign.com (acessado em dez 2012), Fábrica é um dos padrões mais utilizados juntamente com o *Singleton,* podendo ser utilizado em diversas aplicações (Figura 3).

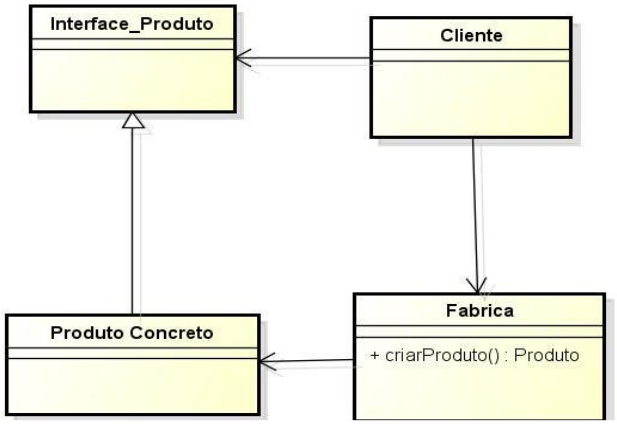

Figura 3 - Modelo Fábrica – Fonte http://www.oodesign.com

Esse modelo cria objetos sem expor a lógica de instalação para o cliente, referindo-se ao objeto recém-criado através de uma interface comum.

3.3.4.4 – Padrão Observador

Baseado no endereço http://www.oodesign.com (acessado em dez 2012), esse *patern* define uma dependência de um-para-muitos entre objetos de modo que enquanto um objeto muda de estado os outros são notificados e atualizados automaticamente (Figura 4).

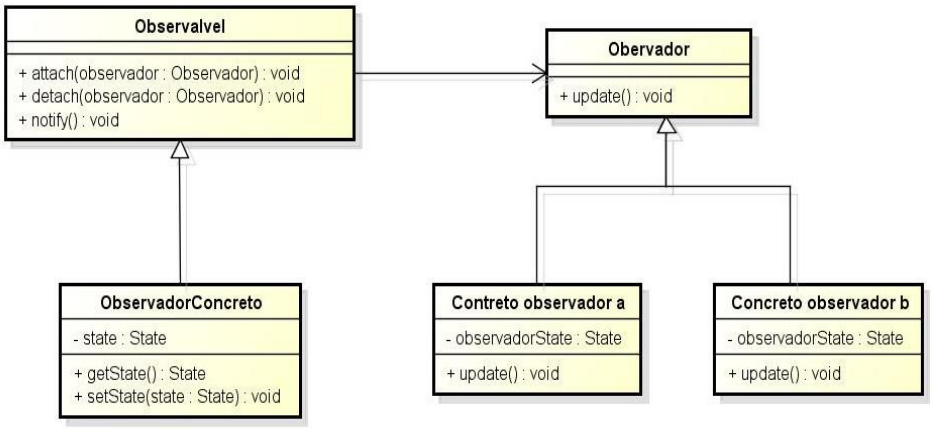

Figura 4 - Modelo Observador – Fonte http://www.oodesign.com

3.3.4.5 – Padrão MVC

O padrão MVC é sem dúvida o modelo de *Designer Pattern* mais reconhecido pelo mercado. Sua principal característica é separar a interface do usuário das demais, formando três camadas de desenvolvimento independentes: *Controller, Model e View.*

Graças a essa característica várias plataformas de desenvolvimento recebem a mesma aplicação modificando apenas a camada de visão, resultando em reuso muito abrangente e diminuindo os custos no projeto (Figura 5).

Magela (2006) define as três camadas da seguinte forma:

- ***Model* –** É a camada de representação de abstração na forma de dados em um sistema.
- ***View* -** É a camada responsável pela interface do usuário.
- ***Controller* –** Camada responsável pelo controle de fluxo do programa, fazendo comunicação entre as camadas *Model* e *View*.

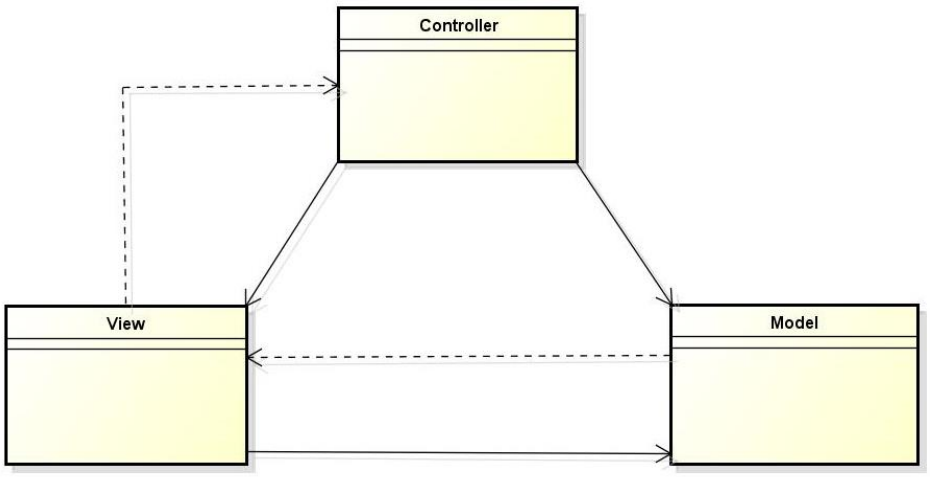

Figura 5 - Modelo MVC – Fonte http://codeigniterbrasil.com

O padrão MVC é muito utilizado como a base para diversos frameworks como, por exemplo, *Cake* PHP, *CODEIGNITER*, *HIBERNATE* etc.

3.3.5 - Reuso de sistemas

A reutilização de sistemas é sem duvida a mais utilizada forma de reuso. Segundo Sommerviller (2007), o reuso de sistema consiste em reaproveitar projetos inteiros de software para reaproveitamento em outros projetos de uso mais específico. O reuso de sistema tem um impacto muito relevante na redução de custos em um projeto de TI, além de diminuir significativamente prazos de entrega.

Sommerviller (2007) acrescenta que existem duas modalidades de reutilização de sistemas amplamente usados no mercado: Reuso de produtos COTS e linhas de produtos de software.

3.3.5.1- COTS

Também conhecido como Produto Comercial, os sistemas COTS[8] são aplicações de uso genérico, usado de forma completa pelo sistema. Atualmente é usado em forma de APIs, onde a sua aplicação é muito utilizada em sistemas de grande porte.

Ciriaco (2009) define a API[9] como um conjunto de padrões de desenvolvimento de sistemas que permite a construção de aplicativos, onde a sua utilização é percebida de forma evidente pelos usuários.

Como exemplo de uso de um sistema de uso COTS, citamos a API do Windows. Através dessa API podemos aproveitar nos projetos de softwares, aplicações como calculadora, agenda, calendário, etc; sem a necessidade de desenvolver essas aplicações, gerando uma grande economia de custos de implementação e tempo de desenvolvimento.

Porém, Sommerviller (2007) alerta que o principal problema apresentado pelo uso de sistemas COTS, especificamente pelas APIs é a sua manutenção. Como as APIs são sistemas externos ao ambiento do projeto, sua manutenção é prejudicada devido ao código-fonte fechado de muitas APIs.

Outro problema apresentado pelas APIs é a sua imprevisibilidade de execução, desempenho e confiabilidade durante a sua incorporação a um projeto de TI.

Se por um lado as APIs proporcionam um ganho de tempo de desenvolvimento, é necessário, porém, testes mais intensos para uma avaliação de viabilidade mais precisa, porém o seu custo beneficio é compensador.

[8] *Commercial Off The Self*
[9] *Application Programming Interface*

3.3.5.2 – Linha de Produtos de Software

Sommerviller (2007) afirma que a linha de produtos de software é sem dúvida uma das metodologias de reuso de sistemas mais eficientes e utilizadas no mercado. A linha de produtos de software consiste em reutilização de um conjunto de regras de negócios de uma aplicação para diversos produtos de software.

Essas regras de negócio comuns para diferentes empresas podem reusados em outros sistemas acrescentados de outros requisitos mais específicos.

Como exemplo, podemos citar um sistema de folha de pagamento. Esse sistema funciona de forma semelhante para todas as empresas que utilizem dele. Porém, nenhuma empresa é igual à outra, cada empresa possui suas características, necessidades para movimentar a folha de pagamento.

Para atender às necessidades mais específicas de cada cliente, basta acrescentar os requisitos específicos ao núcleo, resultando um sistema específico para cada empresa. O benefício dessa abordagem é a não necessidade de desenvolver um sistema de folha de pagamento desde o seu início, bastando somente implementar os requisitos mais específicos.

Sommerviller (2007) mostra que a abordagem de linha de produtos de software pode ser configurada de duas formas:

- **Configuração de tempo de projeto** – onde a empresa desenvolvedora modifica o núcleo do sistema para criar um novo produto para o cliente.
- **Configuração de tempo de implantação** – onde um sistema genérico é desenvolvido com diversos núcleos acoplados. Cada núcleo consiste em um subsistema com suas próprias regras de negócio.

Sommerviller (2007) acrescenta que a configuração de tempo de implantação é a abordagem mais utilizada projetos de software vertical, sendo caracterizado como um sistema ERP ou Sistemas Integrados de Gestão Empresarial.

ERP[10] são sistemas de grande porte com muitos módulos acoplados. Cada módulo da aplicação resulta em um sistema específico para cada necessidade do cliente. Para isso, basta o administrador selecionar quais módulos serão incluídos no produto de software para o cliente.

Os principais problemas apresentados pelos ERPs são: coleta de requisitos, modelagem de processos e tempo de desenvolvimento para cada módulo, podendo ser muito extensos e não ter fim.

Outro problema apresentado pelos ERPs ocorre quando as necessidades de determinados clientes não foi desenvolvido nenhum módulo para esse problema. Nesse caso, há poucos benefícios de reuso de sistema. Muitas vezes ao desenvolver módulos muito específicos para um cliente pode acarretar ineficiência para reaproveitamento de sistema para outros clientes.

Apesar desses problemas, os sistemas ERPs possuem uma grande eficiência e aceitação de clientes e empresas de Software, devido ao seu grande poder de reuso e menor tempo de entrega.

3.4 – Conclusão

É uma tarefa complexa estimar custos de projetos de software de forma transparente, precisa e eficiente para os clientes. Qualquer diferença entre o que foi acordado pelas as partes interessadas e o que foi cobrado de fato podem ser caracterizados como quebra de contrato, podendo inviabilizar o projeto como um todo.

A engenharia de software e o gerenciamento de projetos possuem métricas que ajudam a minimizar esses riscos, ajudando a evitar orçamentos inflados ou mesmo subestimados.

[10] *Enterprise Recourse Planning*

Porém, Yourdon (1995) alerta que fatores externos como licitações com preços fixos pelos governos, saúde financeira da organização comprometida e até mesmo questões políticas podem interferir no sucesso de um projeto de TI.

Para solução desses problemas é fundamental a organização, bom censo e cuidado para não participar de um projeto, chamado por YOURDON (1995), de virtualmente impossível.

4 – Problemas com Prazos de Entrega

Yourdon (1995) afirma que os cronogramas e prazos de entrega de um projeto de TI são sem dúvida as causas mais comuns de fracassos. Cronogramas subestimados e atrasos na entrega do produto final podem resultar no fracasso do projeto como um todo.

Os fracassos decorrentes aos cronogramas são motivados principalmente por pressões orçamentárias, políticas e competitivas. Esses fatores podem não levar em consideração um plano de projeto. Segundo Yourdon (1995), como resultado, os cronogramas são elaborados com a metade do tempo estimado, podendo gerar produtos de software sujeitos a diversos erros, testes insuficientes e ausência de auditorias de software.

Os cronogramas subestimados tem um peso fundamental para fracasso de um projeto de TI. Isso ocorre devido a pressões externas dos clientes. É normal que clientes pressionem que os projetos sejam entregues o mais rápido possível, porém se o gerente de projeto ceder a essas pressões, o projeto pode resultar em fracasso.

Mesmo recorrendo a cronogramas racionais, o projeto de TI pode resultar em fracasso devido aos prazos de entrega não serem cumpridos em contrato. Isso é motivado principalmente por profissionais não capacitados envolvidos, fatores inesperados (problemas com infraestrutura interna, afastamento de colaboradores essenciais ao projeto, etc.) e esforço empregado abaixo do necessário. Como consequência, o produto entregue pode não atender as necessidades da organização e provocar insatisfação de clientes.

No contexto de prazos e entregas de projetos de software, a figura do gerente de projeto é essencial para minimizar os riscos de fracasso, com autonomia para adequar ao projeto, DINSMORE, BREWIN (2009).

Cabe ao gerente de projeto coordenar todos os processos citados acima para garantir que todos os cronogramas sejam os mais racionais possíveis e o projeto seja entregue no prazo estabelecido em contrato. Dinsmore, Brewin (2009),

acrescentam que é de suma importância que o gerente de projeto de TI tenha também bons conhecimentos em engenharia de software.

4.1- Cronogramas de Desenvolvimento de Software

Elaborar um cronograma é sem dúvida uma das formas mais eficientes de se fazer uma estimativa mais realista do prazo de entrega do projeto de TI. O cronograma é elaborado seguindo as seguintes etapas, segundo DINSMORE, BREWIN (2009), definições de atividades, sequenciamento, estimativas de recursos, estimativas de duração das atividades, desenvolvimento e o controle do cronograma.

É importante ressaltar que através de um cronograma bem detalhado é possível também fazer estimativas de custos do projeto, baseando-se nas métricas do capítulo anterior – Custos e Orçamentos nos projetos.

4.1.1- Definições de Atividades

Vargas (2009) define as atividades como etapas necessárias para completar um projeto, onde são executadas em sequência ou mesmo simultaneamente.

Dinsmore, Brewin (2009), acrescentam que as atividades definem as suas necessidades, suas possíveis interdependências e recursos necessários para cumpri-las.

Para facilitar as definições de atividades bem como seu gerenciamento, existem softwares baseados em gráfico *Gantt* e gráficos de rede, por exemplo, MS *Project* e *Gantt Project*. Vargas (2009) acredita que as principais vantagens dessas ferramentas são a visualização de atrasos com maior clareza, melhor entendimento e escalas de tempo melhor definidas. Porém seus principais problemas são inadequações a projetos de grande porte, difícil visualização das dependências das atividades e difícil previsão das alterações de escopo.

Nessa primeira etapa, DINSMORE, BREWIN (2009) afirmam que não é necessário definir datas de início das atividades e nem a equipe estar completamente formada (Figura 6).

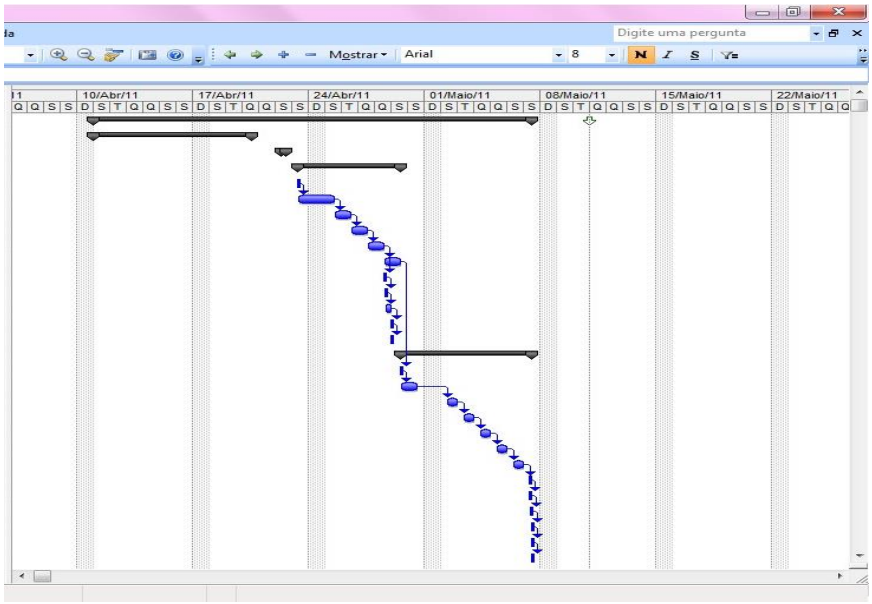

Figura 6 – Gráfico *Gantt* Ms *Project* – Fonte : audyal.blogspot.com

4.1.2 - Sequenciamento de Atividades

Após as definições das atividades, o sequenciamento é a próxima etapa. Consiste em organizar essas atividades em ordem cronológica e principalmente as suas dependências, ou seja, para que uma atividade X seja iniciada, a atividade Y deve ser concluída primeiro.

Segundo Dinsmore, Brewin (2009), o diagrama de rede ajuda a visualizar de forma bem clara as dependências das atividades e quais as que podem ser feitas paralelamente.

Através do diagrama de rede é possível traçar um caminho crítico das dependências de atividades, ou seja, o caminho onde pode haver atrasos no cronograma do projeto por completo.

Vargas (2009) explica que as principais vantagens do diagrama de rede é o simples entendimento e interdependência entre as atividades bem definidas. Porém suas principais desvantagens são as ausências da duração das atividades e dificuldade de manipulação por parte dos usuários. Dinsmore, Brewin (2009)

afirmam que existem softwares que auxiliam a elaboração de diagramas de rede como MS-*Project* (Figura 7).

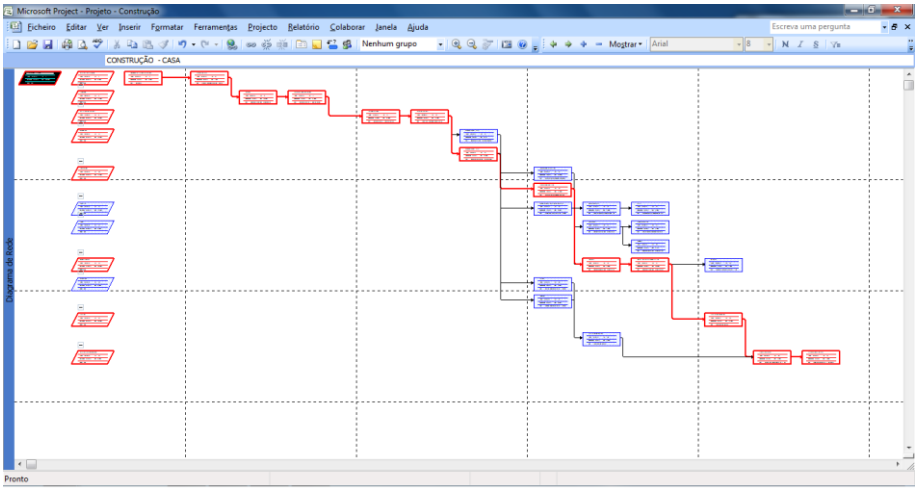

Figura 7 – Diagrama de rede em MS *Project* – Fonte : www.3.bp.blogspot.com

4.1.3 – Estimativa de Recursos para as Atividades

Segundo Dinsmore, Brewin (2009), as estimativas de recursos para as atividades foram acrescentadas ao gerenciamento de tempo somente em 2004 ao guia PMBOK.

Em cada atividade é necessário que o gerente de projeto encontre os recursos humanos e materiais para a sua conclusão com êxito. É importante resaltar que após uma revisão, ainda existam recursos não disponíveis, sendo que é de responsabilidade do gerente de projeto (e não dos líderes de equipe) em garantir o suprimento desses recursos, tomando o cuidado de providenciá-los o mais rapidamente possível interna ou externamente na organização.

Todos os recursos devem ser amplamente documentados, mesmo aqueles que não forem usados em sua totalidade. Dinsmore, Brewin (2009) definem três informações necessárias para se elaborar um cronograma preciso: atividades reais, estimativa de recursos e duração de atividades.

As áreas de gerenciamento de aquisições e gerenciamento de recursos humanos, contidos no guia PMBOK auxiliam o gerente de projeto a definir as estimativas de recursos para atividades com maior precisão.

4.1.4 – Estimativas de Duração das Atividades

Após as atividades estarem definidas e sequenciadas, o gerente do projeto já pode planejar juntamente com a sua equipe as durações de cada atividade.

De acordo com Dinsmore, Brewin (2009), nessa etapa fica mais claro a linha de tempo necessária para o cumprimento das atividades. É uma etapa crítica porque nessa fase o alto comando do projeto começa a cobrar por resultados, porém, o gerente de projeto não possui informações suficientes para uma avaliação precisa de prazo de entrega.

Apesar da ausência de informações mais precisas em relação aos prazos de entrega das atividades, é possível usar as estimativas fornecidas pelas equipes envolvidas no projeto. Essas informações podem ser documentadas e acrescentadas de uma porcentagem de conclusão de cada atividade do cronograma.

Nesse contexto é muito importante o gerente de projeto diferenciar a duração e o esforço empregado para cumprir a atividade. Se uma atividade tiver alocado como recurso um único profissional dedicado apenas 80% do tempo, a sua conclusão não será de dez dias, mas na realidade de 12 dias. Entretanto se dois profissionais estiverem alocados para a mesma atividade esse prazo pode cair para apenas cinco dias.

Geralmente as organizações utilizam suas próprias experiências em projetos passados para buscar uma melhor precisão de suas estimativas, juntamente com o método de estimativa Delphi, que utiliza três parâmetros possíveis de estimativas (provável, otimista e pessimista). Dinsmore, Brewin (2009) definem o método *Button-up* é o mais recomendado, pois pode ser utilizado com outros métodos de estimativas.

Segundo Dinsmore, Brewin (2009) acrescentam que o Software MS *Project*, é uma ferramenta que auxilia na visualização e alocação do esforço empregado e sua duração.

4.1.5 – Desenvolvimento do Cronograma

Dinsmore, Brewin (2009) afirmam que após todas as atividades serem descobertas, sequenciadas, estimadas e documentadas é chegado o momento de desenvolver o cronograma do projeto. Nesse processo o cronograma pode ser atualizado e aperfeiçoado.

Nessa fase as restrições de prazo devem ser confrontadas com as estimativas já documentadas. O gerente de projeto deve renegociar com os líderes de cada processo a adequação às restrições. As folgas dos colaboradores devem estar documentadas, bem como feriados e férias para evitar imprecisões no cronograma.

O grande desafio do desenvolvimento dos cronogramas é adequá-lo ao escopo do projeto. Isso geralmente pode resultar em readequação aos prazos previamente estabelecidos. É sem dúvida o momento mais complexo do desenvolvimento do cronograma devido a essas restrições conflitantes.

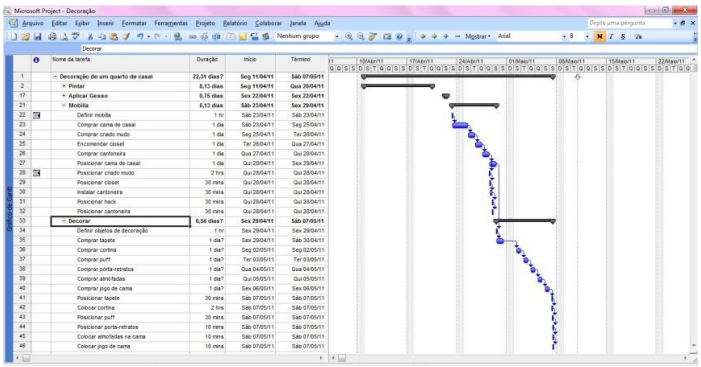

Figura 8 – Cronograma Completo MS Project – Fonte: www.3.bp.blogspot.com

Uma forma de minimizar esses problemas é motivar as equipes a trabalhar interruptamente em suas atividades e procurar convencê-los a se dedicar por um período maior do que já estabelecido. Outra forma de readequação do cronograma

aos prazos é o desenvolvimento das atividades em paralelo, porém os riscos são maiores de retrabalho.

Com o cronograma pronto, o gerente de projeto terá a sua disposição um mapa completo de todo o projeto a ser seguido com maiores possibilidades de conclusão no prazo estabelecido (Figura 8).

4.1.6 - Controle do Cronograma

Segundo Dinsmore, Brewin (2009), nessa última etapa do desenvolvimento dos cronogramas é fundamental para o gerente de projeto, a sua constante manutenção e atualização. É inevitável que uma atividade sofra um atraso ou até mesmo termine antes do prazo estabelecido.

Jenny (2001) lista os principais pilares para um eficiente controle do cronograma:

- Determinação da situação atual do cronograma ao projeto,
- Influência nos fatores que criam mudanças no cronograma,
- Determinação de qual mudança sofreu o cronograma,
- Gerenciamento das mudanças reais no cronograma.

Jenny (2011) acrescenta que existem ferramentas que auxiliam no controle do cronograma. Podemos citar, por exemplo, os softwares de gerenciamento de projeto, análise de desempenho, análise de variação e o nivelamento de recursos.

Segundo Dinsmore, Brewin (2009) afirmam que o maior desafio para um bom controle de cronograma é a obtenção de informações de forma precisa sobre progresso real de uma equipe. Isso ocorre devido às informações não serem suficientemente objetivas.

Outro fator crítico ao controle do cronograma de acordo com HELDMAM (2005) são as mudanças de escopo durante a execução do projeto, essas mudanças

geralmente são propostas por *stakeholders*, clientes e membros da equipe do projeto que identificou outras soluções que não estavam previstas originalmente.

4.2 – Conclusão

Em um projeto de TI o gerenciamento de custos e o gerenciamento de tempo trabalham em conjunto. Se o esforço empregado para cumprir uma atividade de um cronograma for maior do que o previsto, fatalmente subirão os custos de desenvolvimento.

Maximiano (2010), alerta que se o projeto tem mais recursos, o prazo diminui, porém, como consequência os custos subirão e vise-versa, ou seja, se os recursos diminuírem, o prazo de entrega aumenta e como consequência os custos diminuem. Porém os clientes irão pressionar para que projeto seja entregue no prazo, respeitando o orçamento acordado em contrato.

Fica evidenciado que o custo e tempo em projeto de TI são restrições muitas vezes conflitantes, dificultando muito sua conclusão. Cabe ao gerente de projeto uma adequação pertinente onde essas restrições não interfiram um ao outro.

Porém é importante ressaltar na sessão 3.3 - **Reuso para Minimizar os Custos** – contido no capítulo de **Custos e Orçamentos do Projeto**, além de auxiliar na redução de custos, contribui significativamente para redução dos prazos de entrega dos cronogramas de desenvolvimento de software. Isso é possível devido à equação – menor tempo desenvolvimento = > menor custo e vise – versa.

5 – Problemas com a Qualidade do Projeto de TI.

Em um projeto de TI a qualidade, segundo SIEGELAUB (2011), é um conjunto de características do produto entregue ao cliente e o seu nível de tolerância em relação às suas restrições, ou seja, qual será a flexibilidade do cliente em relação às mudanças do projeto.

Como exemplo, em um projeto de TI após uma coleta de requisitos, os engenheiros de software perceberam que alguns requisitos serão impossíveis de ser implementados de acordo com o desejo do cliente. Cabe então aos engenheiros de software negociar com o cliente outras formas de atender esses requisitos sem causar surpresas desagradáveis e principalmente manter o nível de satisfação. SIEGELAUB (2011).

Koscianski e Soares (2007) alertam que os problemas de qualidade de software crescem à medida que o projeto aumenta em complexidade, dificultando muito as renegociações de requisitos com os clientes devido ao grande número de restrições técnicas de implementação.

Além dos problemas referentes à complexidade, KOSCIANSKI, SOARES (2007) descrevem outras situações que afetam substancialmente a qualidade de um software:

- Documentações insuficientes ou incompreensivas para clientes, os tornando difíceis de usar.
- Sistemas que não realizam o esperado pelos usuários.
- Travamentos causados por exceções não tratadas.

Esses problemas podem ser originados por testes insuficientes ou superficiais, falta de treinamento, ineficiência dos usuários e falta de observação ou incompreensão dos requisitos do projeto.

5.1 – A Influência dos Requisitos na Qualidade do Software

Koscianski e Soares (1992, apud CROSBY, 2007) definem a qualidade como ”conformidade aos requisitos”. Essa afirmação exemplifica a importância dos requisitos de um projeto de TI para alcançar a qualidade desejada. A coleta de requisitos é um processo delicado porque é feito no início do projeto, impactando diretamente todas as fases do projeto.

A elicitação de requisitos juntos aos clientes (*stakeholders*) é um processo que pode influenciar de forma decisiva na qualidade de um projeto de TI. Sommerviller (2007) alerta que os *stakeholders* geralmente não sabem realmente quais funcionalidades o software deve possuir. O grande desafio dos engenheiros de software é descobrir através de reuniões as suas reais necessidades.

Outro fator dificultador na elicitação de requisitos, segundo SOMMERVILLER (2007), é o alto nível de abstração no detalhamento de suas necessidades. É possível que um requisito apresentado em linguagem natural por um *stakeholder* gere três ou mais requisitos de sistema.

Se os engenheiros de software não conseguirem interpretar quais são as reais necessidades dos *stakeholders*, poderão ouvir a desagradável afirmação “Não foi isso que pedi”. É uma situação de grande risco, pois os *stakeholders* certamente irão exigir correções do projeto, resultando retrabalho, aumento de custos e prazos.

Uma forma de atenuar esses riscos é coletar requisitos de diferentes *stakeholders* em diferentes níveis da organização, por exemplo, em uma fábrica de móveis de escritório é válido coletar requisitos desde a diretoria até o chão de fábrica. Esse processo ajuda o engenheiro de software a conhecer as reais necessidades da organização.

Porém essa abordagem pode gerar requisitos conflitantes que não podem ser implementados em conjunto. Cabe ao engenheiro de software renegociar com os stakeholders envolvidos quais requisitos deverão ser desenvolvidos.

É importante resaltar que os requisitos devem ser ricamente documentados, segundo convenções próprias ou de outros padrões amplamente reconhecidos, sendo acompanhados em tempo real durante o desenvolvimento.

5.2 – Normas, Framework e Padrões de Qualidade de Software.

Felizmente surgiram nos últimos anos padrões, F*rameworks* e normas de qualidade no desenvolvimento de software que enfocam na busca de melhores práticas de programação, menores custos, menor tempo de desenvolvimento. Podemos destacar, por exemplo, os *Frameworks* COBIT[11] e ITIL[12]. As normas ABNT e ISO. Será apresentado o gerenciamento de qualidade do guia PMBOK e a disciplina de governança de TI.

5.2.1 – ABNT

Koscianski e Soares (2007) afirmam que a ABNT[13] é o principal órgão brasileiro responsável pela qualidade de padrões. A ABNT foi fundada no ano de 1940, sendo também cofundadora da ISO que será analisada na próxima sessão.

A ABNT possui 21 comissões técnicas que normalizam diversos padrões de qualidade em vários setores diferentes como, por exemplo, mineração e construção civil. Na área de engenharia de software a ABNT possui sete comissões de estudo onde podemos destacar as comissões de qualidade de software, gerência do ciclo de vida e estimativa de tamanho de software (tabela 2).

CE	Área de Estudo
21:101.01	Qualidade de Software
21:101.03	Gerencia do Ciclo de Vida
21:101.06	Estimativa do Tamanho de Software
21:101.08	Ergonomia

Tabela 5 - Algumas Comissões da ABNT – fonte Koscianski e Soares (2007)

[11] *Control Objectives for Information and Related Technology*

5.2.2 – ISO

Segundo Koscianski e Soares (2007), o padrão ISO surgiu no final da segunda guerra mundial na cidade de Londres com a colaboração de 25 países, porém a sua sede está localizada em Genebra – Suíça. O padrão ISO utilizou várias métricas de outros padrões já existentes, como por exemplo, ISA[14].

O termo “ISO” não é uma sigla, porém é derivado de uma palavra grega “isos” que significa “igual”. No passado o ISO se dedicava apenas em elaborar documentos de recomendações. Hoje são mais de 15.000 padronizações de qualidade reconhecidas internacionalmente.

Koscianski e Soares (2007) acrescentam que as normas ISO 9000 tornou-se símbolo de preocupação com a qualidade. Depois disso surgiu a norma ISO/IEC 15504 que avalia os processos nas organizações.

Após a boa aceitação da norma ISO/IEC 15504, foi verificada a necessidade de uma nova norma voltada exclusivamente para a qualidade de software. Essa norma foi batizada de SPICE Network, sendo composta de cinco partes, conforme exibido na tabela 3.

Parte	Descrição
1	Conceitos e vocabulário
2	*Framework* do processo de avaliação
3	Recomendações para realização de uma avaliação
4	Recomendações para melhora dos processos
5	Contem um exemplo de aplicação

Tabela 6 - Divisão da Norma ISO/IEC 15504 Fonte : Koscianski e Soares (2007)

É importante resaltar que ao contrário do termo “norma”, não se aplica ao SPICE Network, devendo ser usado em conjunto com norma ISO /IEC 12207 que se preocupa principalmente com os processos do ciclo de vida de um projeto.

[12] *Information Technology Infrastructure Library*
[13] Associação Brasileira de Normas Técnicas
[14] *International Federation of the National Standardizing Associations*

5.2.3 – Gerenciamento de Qualidade do Guia PMBOK

Segundo Vargas (2009), o guia PMBOK, publicado pelo PMI possui nove áreas de conhecimento que auxiliam no gerenciamento de projeto. Uma dessas áreas é o gerenciamento de qualidade que será detalhado nessa sessão.

Vargas (2009) acrescenta o gerenciamento de qualidade é subdivido em três áreas: Planejar a qualidade, rastrear a garantia de qualidade e garantir o controle de qualidade. Esses três processos possuem suas próprias entradas, ferramentas e as suas saídas.

5.2.3.1 – Planejar a Qualidade

Vargas (2009) afirma que o planejamento da qualidade, consiste na análise dos requisitos do projeto e seus padrões, além de verificações constantes de suas documentações, possuindo suas entradas, ferramentas e saídas (Tabela 4).

Entradas	Ferramentas	Saídas
• Linha de base do escopo. • Registro das partes interessadas. • Linha de base do desempenho do custo. • Linha de base do cronograma. • Registro dos riscos. • Fatores ambientais da empresa e ativos de processos organizacionais.	• Análise de custo-benefício. • Custo da qualidade, gráficos de controle. • Projeto de experimentos. • Amostragens estatísticas. • Fluxogramas • Metodologias de qualidade • Ferramentas de planejamento de qualidade.	• Plano de gerenciamento de qualidade. • Métricas de qualidade • Listas de verificação de qualidade. • Plano de melhoria nos processos. • Atualizações nos documentos do projeto.

Tabela 7 - Entradas, Ferramentas e Saídas Planejamento Qualidade.

5.2.3.2 – Realizar a Garantia de Qualidade

Vargas (2009) explica que uma auditoria é realizada para verificar se os requisitos estão sendo cumpridos e se as medições de controle estão usando os

padrões de qualidade requeridos e possuindo suas entradas, ferramentas e saídas (Tabela 5).

Entradas	Ferramentas	Saídas
• Plano de gerenciamento de projeto • Métricas de qualidade • Informações sobre o desempenho do trabalho e medições do controle de qualidade.	• Técnicas de planejamento e qualidade • Auditoria de qualidade • Análise de qualidade	• Atualizações dos ativos de processos organizacionais • Solicitações de mudança • Atualizações do plano de gerenciamento do projeto e atualizações dos documentos do projeto.

Tabela 8- Entradas, Ferramentas e Saídas da Garantia de Qualidade.

5.2.3.3 – Realizar o Controle de Qualidade

De acordo com Vargas (2009), o controle de qualidade é o processo que monitora os resultados da qualidade na execução das atividades de um projeto. O controle de qualidade ajuda a avaliar o desempenho, recomendando mudanças se necessário. O controle de qualidade possui suas próprias entradas, ferramentas e saídas (tabela 6).

Entradas	Ferramentas	Saídas
• Plano de gerenciamento do projeto. • Métricas da qualidade • Listas de verificação de qualidade • Medições do desempenho do trabalho • Solicitações de mudança aprovadas • Entregas e ativos de processos organizacionais.	• Diagrama de causa e efeito • Gráficos de controle • Fluxogramas • Histograma • Diagrama de Pareto • Gráfico de execução • Inspeção e revisão de solicitações de mudanças aprovadas.	• Medições do controle da qualidade. • Mudanças validadas. Entregas validadas. • Atualizações dos processos organizacionais. • Solicitações de mudança. • Atualizações do plano de gerenciamento do projeto. • Atualizações dos documentos do projeto.

Tabela 9 Entradas, Ferramentas e Saídas Controle de Qualidade

5.2.4 - COBIT *Framework*

Fagundes (2013) define o *Framework* COBIT como um guia, publicado pelo ISACF[15] que auxilia no gerenciamento dos processos baseado nas regras de negócios. Por ser orientado a eventos, o COBIT é amplamente reconhecido por profissionais de TI, gerentes de projeto e auditores como uma ferramenta de controle de qualidade de software.

Entre os benefícios do COBIT podemos citar, por exemplo, a independência de plataforma de TI, medições dos processos e gerenciamento de desempenho. O COBIT é dividido em quatro domínios e subdividido em 34 processos conforme figura 9. Muitos desses processos são derivados de outros modelos de boas práticas de gestão como, por exemplo, PMBOK, ISO/IEC, ITIL, etc. Sua contribuição é fundamental para uma boa governança de TI (Figura 9).

Recentemente o COBIT foi atualizado para versão 5.0, porém a versão 4.1 é a versão traduzida para o português e comumente conhecida.

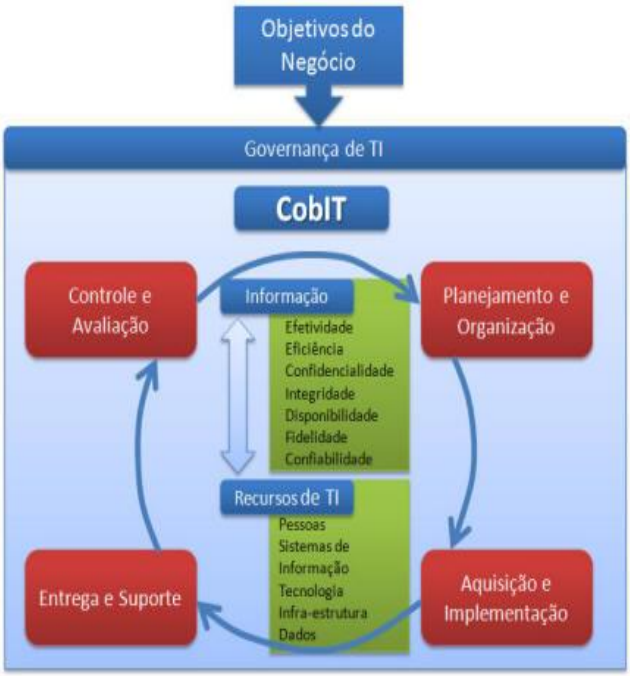

Figura 9 - Os quatro domínios do COBIT – Fonte: www.efagundes.com/artigos/cobit.htm

[15] *Information Systems Audit and Control*

5.2.5 – ITIL

Segundo Mansur (2009), o ITIL[16] sendo originalmente criado pela secretaria de comércio inglês é o principal modelo de referência para gerenciamento de TI com maior reconhecimento e aceito mundialmente.

Por tudo isso o ITIL é considerado um dos principais *Frameworks* de qualidade de software. Atualmente o método ITIL tornou-se norma BS-15000 sendo um anexo da ISO 9000/2000. Seus principais módulos são: transição de serviços, desenho de serviços e operação de serviço, onde todos esses módulos comunicam-se mutuamente (figura 10).

Figura 10 – Estrutura ITIL – Fonte – www.profissionaisdatecnologia.com.br

Mansur (2009) acrescenta que o ITIL tem como principais objetivos as melhores práticas de programação independente da plataforma usada, ou seja, não é restrito pelas linguagens de programação ou banco de dados, sendo muito flexível neste ponto.

Outras características apontadas por Mansur (2009) sobre o ITIL são: modelo de referência para processos não proprietários, sendo adequado para todas as áreas de atividades, *checklists* de testes e restrições para atividades que não podem ser realizadas.

[16] *Information Technology Infrastructure*

O modelo ITIL produz os seguintes resultados (MANSUR, 2009): fortalecimento dos controles e da gestão dos ambientes de TI, redução do tempo de desenvolvimento dos processos, diminuição dos recursos e sistemas de TI, satisfação de usuários e clientes, redução de custos em TI, extensão do ciclo de vida dos processos e maior reconhecimento de capacidade de colaboradores e clientes.

5.3 - Governança de TI

A Governança de TI, afirma WEILL (2004), é um dos braços da governança corporativa sendo importante a sua compreensão. Devido a muitos escândalos de grandes corporações no EUA, a confiança de muitos investidores ficou comprometida resultando, com isso, queda nas ações. A governança corporativa tem como um dos seus objetivos oferecer informações transparentes, independentes e precisas dos seus balanços aos seus investidores.

A Governança de TI, acrescenta WEILL (2004), coloca a TI no centro dos processos decisórios para estimular o uso de seus comportamentos, ou seja, aponta quem e quanto serão investidos em TI.

Dorow (2010) afirma que o papel da governança de TI é garantir que as informações sejam fidedignas através de controles de forma mais transparente possível para stakeholders.

Weill (2004) lista os cinco principais conceitos de governança de TI auxiliam na compreensão de sua importância nos processos decisórios nos negócios:

- Princípios de TI: qual é o papel da TI no negócio.
- Arquitetura de TI: quais são os requisitos de integração e padronização.
- Infraestrutura de TI: quais os serviços compartilhados e de suporte.
- Necessidades e aplicações de negócio: quais são as necessidades e aplicações de TI no negócio.
- Investimento e prioridades em TI: onde e como investir em TI.

Weill (2004) acrescenta que essas cinco decisões são interligadas e vinculadas para uma melhor governança de TI. Os resultados de uma boa governança de TI são:

- Estratégias de negócio mais claras e o papel da TI nesse contexto.
- Controle do gerenciamento de gastos e lucros com a TI.
- Desenvolvimento das atividades mais ágeis.

5.4 - Conclusão

A qualidade é um fator crítico para o sucesso de um projeto de TI. Por isso, recentemente foi inserida por diversos profissionais de gerenciamento de projeto, a chamada “tríplice restrição” - escopo, tempo e custo.

Existem outros padrões, *Frameworks* e normas de qualidade além dos mencionados nesse trabalho. Podemos citar o CMM[17] - que auxilia nas melhores práticas de programação; o PPM[18] - que auxilia na gestão dos processos; Fábrica de Software - onde o desenvolvimento é construído como uma linha de montagem, agilizando todo o processo.

Koscianski e Soares (2007) citam um exemplo simples para comprovar a dificuldade de gerenciar a qualidade de um projeto de TI: uma lâmpada tem em suas especificações o consumo 60 w de potência, mas qual a garantia que o consumo é exatamente esse ou de 59,8W ou 60,1W? Em um projeto de TI essas diferenças podem ser mais evidentes na avaliação da qualidade.

[17] *Capability Maturity Model*
[18] *Project Portfolio Management*

6 – Problemas com o Escopo do Projeto

Dinsmore, Brewin (2009) afirma que o escopo é sem dúvida o principal fator crítico de sucesso de um projeto de TI, juntamente com as restrições de prazo e custo. Formando a chamada “tríplice restrição” de um projeto.

As restrições do escopo se assemelham com o gerenciamento de qualidade, entretanto, SIEGELAUB (2011) descreve que a principal diferença é sua entrega que está mais focada no produto entregue, ou seja, não há limites de aceitação por parte dos clientes.

Como exemplo citado por Siegelaub (2011), ao entregar um produto, o cliente e o gerente de projeto verificam se o prazo e o custo estão dentro dos termos estabelecidos para uma possível entrega de outro produto. Isso é denominado de “Faixa de Variação de Valores”. Se não for possível, não será caracterizado em um fracasso do projeto.

Paiva (2008) cita alguns problemas de escopo que pode resultar em fracasso em um projeto de TI:

- **Escopo incompleto.** - Ocorre quando um sistema não é decomposto em subsistemas de forma correta, ou seja, um subproduto importante do sistema não será incluído podendo comprometer todo o projeto.
- **Nível de detalhamento inadequado ou insuficiente.** Mesmo com escopo completo, seus subsistemas não apresentam um correto detalhamento, prejudicando estimativas de tempo e custos.
- **Documentação de escopo incompreensível para a equipe envolvida.** Ausência de um dicionário analítico pode causar confusão para colaboradores recém-contratados no projeto podendo resultar em interpretações ambíguas.

Segundo Dinsmore, Brewin (2009), os produtos gerados pelo escopo do projeto possuem importância fundamental para o sucesso de um projeto de TI como um todo.

6.1 - Produtos do Gerenciamento do Escopo

As etapas para se obtiver produtos de escopo com grande valor agregado são: Linha de Base do Escopo, Coleta de Requisitos das Partes Interessadas, Definição do Escopo, Criação do Escopo, Verificação do Escopo e Controle do Escopo.

6.1.1 - Linha de Base do Escopo

Segundo Dinsmore, Brewin (2009), a linha de base do escopo deve conter os processos e metodologias iniciais da documentação do escopo de um projeto de TI. Geralmente seus processos envolvem as seguintes bases:

- Processo para coleta de requisitos.
- Processo para validação de requisitos (aceitação do cliente).
- Processo para documentar os requisitos (definição do escopo).
- Processo para identificação das entregas do projeto**.**
- Processo de verificação e controle do escopo.
- Processo para documentação das métricas de controle do escopo.
- Processo para estruturação de equipes envolvidas.

Dinsmore, Brewin (2009), afirmam que esses processos podem variar de acordo com as necessidades do projeto, porém devem ser documentadas para futuras revisões da equipe envolvida.

6.1.2 – Coleta de Requisitos

Conforme visto na seção 5.1, os requisitos são fundamentais para a qualidade de um projeto. Para o seu sucesso é fundamental, DINSMORE, BREWIN (2009), explicam que a listagem das partes interessadas (*Stakeholders)* diretamente ligados com as regras de negócio.

Os resultados das reuniões com as partes interessadas deverão ser elaboradas em um documento de requisitos. Percebendo a importância dos *Stakeholders* na coleta dos requisitos de um projeto, o PMI criou uma nova área de conhecimento do guia PMBOK – Gerenciamento de Stakeholders.

6.1.3 - Definição de Escopo

Segundo Dinsmore, Brewin (2009), após a coleta de requisitos, o escopo passa pela etapa de definição que é o seu detalhamento. Esta é uma etapa crítica porque se o nível de detalhamento do projeto se estender ou não cumprir os requisitos mínimos de qualidade, o projeto de TI resultará em fracasso.

É fundamental que a definição do escopo tenha objetivos claros, restrições bem definidas e quais os requisitos que não podem fazer parte do escopo. Dinsmore, Brewin (2009) acrescentam outros documentos como: portfólios de investimento, orçamentos e análise de mercado podem ajudar na definição do escopo.

Dinsmore, Brewin (2009) acreditam que se a definição do escopo for bem detalhada, todas as partes envolvidas no projeto compreenderão com clareza suas entregas, saídas e seus objetivos. Quaisquer mudanças da definição do escopo devem ser aprovadas pelo gerente de projeto.

6.1.4 – Elaboração da Estrutura Analítica do Projeto.

Nessa etapa, Dinsmore, Brewin (2009) afirmam que a definição do escopo é decomposta em partes menores, que se tornarão os produtos do projeto. Cada produto será entregue a equipe competente, detalhando prazos de entrega, custos e aquisições.

É possível perceber nesse contexto que as outras áreas do gerenciamento de projetos como: gerenciamento de integração, gerenciamento de tempo, gerenciamento de custos, gerenciamento de aquisições e gerenciamento de recursos humanos, tem importância no cumprimento do escopo.

Cada produto contém seu próprio escopo, sendo muito importante distinguir a diferença entre escopo do projeto e escopo do produto. Para Maximiano (2010), em uma estrutura analítica do projeto existem pelo menos três entregas: o produto, o capital humano e a infraestrutura necessária para o seu correto funcionamento.

6.1.5 – Aprovação do Escopo.

Segundo Dinsmore, Brewin (2009), após a estrutura analítica do projeto ser concluída, o documento do escopo é apresentado pelo gerente do projeto para todas as partes interessadas (equipe, financiador, clientes), visando a aprovação, esclarecimento de dúvidas ou mesmo modificação de partes do escopo.

Essa é a última oportunidade de modificação de partes do escopo sem prejuízo de custos e tempo para o projeto. Se o escopo for modificado, durante o desenvolvimento das atividades os riscos de fracasso aumentam consideravelmente.

6.1.6 - Verificação e Controle do Escopo

Dinsmore, Brewin (2009) acrescentam que a última etapa da criação dos produtos do escopo do projeto começa quando as entregas do projeto são finalizadas. A responsabilidade do gerente de projeto é garantir que o escopo seja cumprido em sua integridade, não permitindo que os produtos gerados saiam da declaração do escopo.

A verificação e controle do escopo trabalham juntamente com gerenciamento de qualidade, pois é mensurado a partir dos requisitos do produto, onde é gerenciado por todo ciclo de vida do projeto.

Para Maximiano (2010), para auxiliar o controle do escopo, os produtos devem responder as seguintes perguntas:

- Como o projeto esta sendo realizado?
- De que forma o projeto esta sendo realizado?
- Que modificações os clientes e outras partes interessadas propõem?

Maximiano (2010) acrescenta que as mudanças no escopo nessa etapa, apesar de indesejáveis, são muito frequentes impactando diretamente no custo e no prazo de entrega. Portando é prudente prever quais as mudanças que os clientes poderão propor.

6.2 – Documento de Declaração do Escopo.

Segundo Dinsmore, Brewin (2009), após a finalização das definições dos produtos gerados pelo escopo do projeto, será construído o documento denominado declaração do escopo. Esse documento conterá toda a descrição do escopo do projeto, escrito em linguagem natural, possibilitando que todas as partes interessadas tenham um claro entendimento.

Coimbra (2012) afirma que uma declaração de escopo bem construída é a base de projeto bem sucedido. Esse documento contribui para um melhor esclarecimento para as partes interessadas das entregas dos produtos, suas restrições e possíveis mudanças.

Para Dinsmore, Brewin (2009), uma declaração de escopo bem documentada, trará benefícios imediatos para o projeto, as partes interessadas terão menos questionamentos e dúvidas sobre o escopo. Existem seis indicadores da declaração de escopo que impactam diretamente no sucesso de um projeto:

- Integração de processos e produtos.
- Tipo de ambiente e cultura na execução dos processos.
- Comprometimento das partes interessadas.
- Treinamento e educação.
- Formalização das metodologias.
- Melhor aproveitamento dos recursos disponíveis.

O documento de declaração de escopo deve conter de acordo com Coimbra (2012), uma visão geral do projeto, resumos das prestações de contas e suas entregas, objetivos do projeto em relação às restrições (prazo, custo e qualidade) e uma descrição detalhada de como o projeto será gerenciado. Entretanto esse documento não poderá conter parâmetros externos do escopo como, por exemplo, contratos de pagamento, referências pessoais, e cronograma de pagamentos de honorários da equipe que está construindo o projeto. Essas informações devem ser detalhadas em outros documentos apropriados.

6.3- Conclusão

O gerenciamento do escopo de projeto de TI, trabalha juntamente com o gerenciamento de qualidade, sendo considerado um dos elementos da “Tríplice Restrição do Projeto” (prazo, custo, escopo).

Um escopo muito extenso pode acarretar problemas como: aumento de custos, dificuldade de cumprir prazos e não conformidade com as necessidades das partes interessadas. Porém, um escopo muito restrito acarretará em risco de produtos importantes não serem entregues, resultando em fracasso. Portando o grande desafio de um escopo bem sucedido é a sua adequação aos prazos mínimos e aos custos reduzidos.

Como exemplo dos problemas do escopo, podemos fazer uma analogia a um projeto de um parque de diversões. Se nesse projeto constar como subproduto uma “piscina olímpica”, fica caracterizado como um escopo desnecessariamente extenso. Porém se nesse projeto não constar o subproduto “carrossel”, fica caracterizado um escopo restrito. É fundamental que o gerente de projeto encontre o equilíbrio do escopo.

A comunicação com as partes interessadas (*Stakeholders*) é fundamental para a criação de uma declaração de escopo eficiente. Com o intuito de aprimorar essa comunicação, o PMI[19] atualizou o guia PMBOK para a versão 5.0, adicionando uma nova área de conhecimento: O gerenciamento voltado para os *Skakeholders*. Segundo Trentin (2013), o gerenciamento de *Stakeholders* possui quatro processos que auxiliam em um melhor relacionamento com essas partes interessadas e consequentemente auxiliar na definição do escopo. Seus processos são:

- Identificar *Stakeholders.*
- Planejar gerenciamento de *Stakeholders.*
- Gerenciar engajamento de *Stakeholders.*
- Controlar engajamento dos *Stakeholders.*

[19] *Institute Management Project*

7 – Considerações Finais

Por que a maioria dos projetos de TI resulta em fracasso? Como converte-los em sucesso? Ao longo desse trabalho descobrimos que a “Tríplice Restrição”, ou seja, custo, prazo e escopo somando a qualidade possui um impacto direto na tênue linha entre o fracasso e o sucesso.

Porém, não é possível garantir o sucesso de um projeto de TI somente tomando cuidado nessas restrições. As abordagens estudadas nesse trabalho contribuem para minimizar os riscos, porém não significa a sua eliminação completa. Outras restrições do projeto podem impactar diretamente em seu fracasso, como por exemplo, fatores externos (políticos, concorrência com outras organizações, etc.).

Fatores culturais também são decisivos para o fracasso de um projeto de TI. Muitas empresas recusam-se a adotar as melhores práticas de Engenharia de Software, Gerenciamento de Projetos e Governança de TI; acreditando somente que a experiência no mercado, infraestrutura e boa equipe de colaboradores são suficientes para o seu sucesso.

Como exemplo real de fracasso de projeto de TI, podemos citar a distribuidora de medicamentos americana *Foxmeyer Drug* que declarou falência devido ao projeto de TI encomendado a SAP e *Andersen Consulting* (hoje *Accenture*).

Entretanto podemos citar o software da NASA que controla as missões espaciais americanas que utiliza apenas 1/100 de linhas de código em relação ao *Windows* XP e não é alterado há mais de vinte anos, sendo considerado o software melhor projetado até hoje (VIEIRA, 2003).

Podemos concluir como provável que o tratamento dessas restrições contribui de forma significativa para minimizar os riscos de fracasso de um projeto de TI.

REFERÊNCIAS.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **Citação**: NBR-10520/ago - 2002. Rio de Janeiro: ABNT, 2002.

_____. **Referências**: NBR-6023/ago. 2002. Rio de Janeiro: ABNT, 2002.

DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar**. São Paulo: Editora Pearson Education, 8ª Ed. 2010.

DINSMORE, Paul C., Cabanis – Brewin, Jannette **AMA - Manual de Gerenciamento de Projetos** Rio de Janeiro – Brasport 2009.

KOSCIANSKI, André; Soares, Michel dos Santos. **Qualidade de Software** São Paulo,2ª Ed.2006.

MAGELA, Rogério. **Engenharia de Software Aplicada**. Rio de Janeiro: Alta Boocks, 2006.

MARTINS Junior, Joaquim. **Como Escrever Trabalhos de Conclusão de Curso**. São Paulo: Vozes 5ª ed. 2008.

MAXIMIANO, Antonio Cezar Amaru. **Administração de Projetos, Como Transformar Ideias em Resultados** – São Paulo Atlas 4ª ed. 2010.

MEIRA, Silvio Lemos. **Um Mundo Feito (quase completamente) de Software**. Cienc. Cult. [online]. 2003, vol.55, n.2, pp. 24-28. ISSN 0009-6725.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo - Makron Books, 3ª Ed. 1995.

SIEGELAUB, Jay M. (artigo) **Revista Mundo Project Maneger** – ano 06 - 2011 n.º 33 pags. 22-27.

SOMMERILLE, Ian. **Engenharia de Software**. São Paulo: Pearson Addison-Wesley, 8 ª ed., 2007.

VARGAS, Ricardo Vargas. **Manual Prático do Plano de Projeto Utilizando o PMBOK GUIDE**. 4ª ed. Rio de Janeiro: editora Brasport, 2009.

WEILL, Peter – Ross, Jeanne W. **Governança de TI** São Paulo: Makron Books 2004

YOURDON, Edward. **Projetos Virtualmente Impossíveis** São Paulo – Makron Books, 1996.
http://www.oodesign.com (acessado em dez 2012)

http:tecnomundo.com.br/programacão(acessado em mar 2013)

www.houaiss.uol.com.br (acessado em abril 2013)

www.milhelis.uol.com.br (acessado em abril 2013)

www.aulete.uol.com.br (acessado em abril 2013)

Fagundes, Eduardo Mayer (artigo)-www.efagundes.com/artigos/cobit.htm (acessado em abril 2013).

Jenny, Juliana - julianakolb.com (acessado em abril 2013)

Mansur, Ricardo – www.profissionaisdatecnologia.com.br (acessado em abril 2013)

Fabio, Luiz Henrique de Paiva – www.ogerente.com/stakeholder (acessado em maio 2013).

Coimbra, Rodrigo – www.projetosti.com.br/gestao (acessado em maio 2013).

Trentim, Mario H. – www.brog.mundopm.com.br (acessado em maio 2013).

Alecrim, Emerson – www.infowester.com/ti (acessado em maio 2013).