UCAM
UNIVERSIDADE CANDIDO MENDES

PÓS-GRADUAÇÃO *LATO SENSU* EM TECNOLOGIA DA INFORMAÇÃO E ANÁLISE DE NEGÓCIOS

IMPLANTAÇÃO DE UMA SISTEMÁTICA DE MÉTRICAS

A Difícil Arte de Estimar Tempo para Implementação de Sistemas de informação

Monografia apresentada em cumprimento às exigências para obtenção de grau no curso de pós-graduação *lato sensu* de especialização em Tecnologia da Informação na Administração de Negócios.

Por:
Carlos Simões

Rio de Janeiro, 15 de Maio de 2004.

DEDICATÓRIA

Ao meu filho Lucas que é a razão de minha vida, meu porto seguro.

UCAM
UNIVERSIDADE CANDIDO MENDES

AGRADECIMENTOS

Aos meus caros professores e companheiros de trabalho que tanto me apoiaram e me forneceram informações para acumular conhecimento na difícil tarefa de estimativa de sistemas de informação. Em especial a Aguinaldo Aragon Fernandes que foi a primeira pessoa a me transmitir informações sobre métricas.

EPÍGRAFE

"Desenvolver sistemas de informação sem procedimentos padronizados e sem um método de estimativa é o mesmo que se aventurar em um mar tenebroso sem instrumentos de navegação".

O Abismo da Borda do Mundo:

Embora seu país fosse banhado pelo Atlântico, os portugueses nunca haviam desafiado o Mar Tenebroso, um território mitológico e desconhecido.

Os próprios árabes acreditavam que as "portas" daquele oceano eram guardadas, com a ajuda de um dragão, pelas "ninfas do Poente", as Hespérides, filhas do gigante Atlas.

Disposto a chegar à Guiné - de onde provinha o ouro que enriquecera Ceuta - D. Henrique decidiu enfrentar os perigos do Atlântico.

Os marujos, de Sagres desesperaram-se, pois acharam que seus navios iriam despencar no abismo do fim do mundo: a maioria deles acreditava que a Terra era plana como uma bandeja.

Para além desses terrores imaginários, eles tiveram que enfrentar uma série de perigos reais: os ventos e as correntes contrárias, as longas calmarias sob o sol inclemente e as súbitas tempestades.

Os mareantes, além disso, deparavam estranhos fenômenos meteorológicos - como o chamado 'fogo de São Telmo "(correntes elétricas que atingiam os mastros) - e com incríveis animais marinhos: peixes imensos, enormes polvos, baleias descomunais".

Tais seres deram origem ao mito das sereias e das serpentes marinhas - sempre prontas, na febril imaginação dos marujos, a devorar seus navios.

Livro de Eduardo Bueno – Brasil Terra a Vista [29].

RESUMO

Uma das maiores dificuldades encontradas no gerenciamento de projetos de informática é saber a dimensão do que está sendo gerenciado. Muitas aplicações que a princípio parecem pequenas, quando em desenvolvimento, mostram-se muitas vezes maior do que o previsto inicialmente e, para alguns casos, torna-se tão complexas e grandes, que se perde o controle. Além do que nem sempre é viável se lançar na aventura de desenvolver um aplicativo, já que atualmente existem no mercado uma infinidade de produtos prontos ou necessitando de pequenas customizações.

A área de informática da grande maioria das empresas, em geral, só consegue retratar custos passados de hardware, software e *peopleware*, além de alguns indicadores relativos ao ambiente de produção (utilização de processador por tempo, taxa de disponibilidade do sistema, etc.). A inexistência de indicadores de desempenho financeiro, de qualidade e de produtividade que retratem o desenvolvimento de sistemas de informação dificulta, e muito, a efetiva gerência destas atividades. Produzir serviços de alta qualidade com o mínimo custo possível - ou seja, alta produtividade - se constitui em fator crítico de sucesso para o bom desempenho empresarial.

No esforço de implantação de uma nova mentalidade voltada para a qualidade e produtividade, não é admissível o desenvolvimento de sistemas através de 'feeling', onde questões como: qual é a produtividade da área de informática, qual é a capacidade de produção, qual conjunto de ferramentas possibilita a maior produtividade, quais são os indicadores de qualidade existentes, qual caminho devo adotar, desenvolver ou comprar um pacote e customizá-lo, são simplesmente deixadas de lado ou respondidas sem suporte de uma base quantificável.

Um processo de medição deve ser implantado na empresa com o objetivo de suportar a análise de tendências de determinados indicadores, que podem subsidiar ações para reversão ou sustentação dessas tendências; a análise de impactos na introdução de novas tecnologias sobre a qualidade e produtividade, que pode auxiliar na decisão sobre quais combinações de elementos de tecnologia garantem melhores resultados; a análise de atributos, que permite a comparação da qualidade e

produtividade entre plataformas, metodologias, áreas de aplicação, habilidades técnicas de pessoas e assim sucessivamente.

A gerência, criando os padrões de medidas para a instalação, fica em posição de monitorar o comportamento dos projetos e produtos individualmente, analisar resultados, compará-los e verificar a adequabilidade dos respectivos processos e necessidades de implementar melhorias.

SUMÁRIO

INTRODUÇÃO

Uma das maiores dificuldades encontradas no gerenciamento de projetos de informática é saber a dimensão do que está sendo gerenciado. Muitas aplicações que a princípio parecem pequenas, quando em desenvolvimento, mostram-se muitas vezes maior do que o previsto inicialmente e, para alguns casos, tornam-se tão complexas e grandes, que se perde o controle. Além do que nem sempre é viável se lançar na aventura de desenvolver um aplicativo, já que atualmente existem no mercado uma infinidade de produtos prontos ou necessitando de pequenas customizações.

Um sem-número de dúvidas vem à tona, quando se fala em dimensionamento, prazo e custo de sistemas. Estas dúvidas não são só pertinentes aos gerentes e desenvolvedores e sim, a todos os envolvidos neste processo tão nebuloso e complicado de se administrar. Dentre elas, pode-se citar algumas, como por exemplo:

- Fornecer expectativas realistas para o usuário / cliente;
- Avaliar e medir resultados;
- Ter conhecimento do patrimônio de software;
- Obter reconhecimento relativo a um bom trabalho;
- Estimativa de prazo, custo e recursos para desenvolver sistema ou customizar pacote;
- Decidir entre manter x desenvolver x comprar;
- Estabelecer indicadores para tomada de decisão;
- Participar do processo de qualidade.

A área de informática da grande maioria das empresas, em geral, só consegue retratar custos passados de hardware, software e *peopleware*, além de alguns indicadores relativos ao ambiente de produção (utilização de processador por tempo, taxa de disponibilidade do sistema, etc.).

A inexistência de indicadores de desempenho financeiro, qualidade e produtividade que retratem o desenvolvimento de sistemas de informação dificulta, e muito, a efetiva gerência destas atividades. Produzir serviços de alta qualidade com o

mínimo custo possível - ou seja, alta produtividade - se constitui em fator crítico de sucesso para o bom desempenho empresarial.

A gestão do ambiente de software não está vinculada somente a um projeto ou a um produto específico e sim, ao conjunto dos projetos e produtos da instalação como um todo. É o nível tático de gestão. Neste nível, a preocupação é avaliar a qualidade dos processos de planejamento de projetos, de desenvolvimento de software e de gestão dos produtos em utilização, visando atingir patamares cada vez mais elevados de qualidade sob o conceito de melhoria contínua.

Para tanto, as medições operacionais devem ser agregadas a fim de permitir: a análise de tendências de determinados indicadores, que podem subsidiar ações para reversão ou sustentação dessas tendências; a análise de impactos na introdução de novas tecnologias sobre a qualidade e produtividade, que pode auxiliar na decisão sobre quais combinações de elementos de tecnologia garantem melhores resultados; a análise de atributos, que permite a comparação da qualidade e produtividade entre plataformas, metodologias, áreas de aplicação, habilidades técnicas de pessoas e assim sucessivamente.

Uma das dimensões da gestão do software é a econômica. O esforço de atingir patamares mais evoluídos de qualidade recai também sobre a gestão de custos de não-conformidade ou má qualidade. Ou seja, qualidade aumenta lucratividade pela diminuição dos custos de falhas internas e externas e pelo aumento da satisfação do cliente.

A gerência, criando os padrões de medidas para a instalação, fica em posição de monitorar o comportamento dos projetos e produtos individualmente, analisar resultados, compará-los e verificar a adequabilidade dos respectivos processos e necessidades de implementar melhorias.

1. Premissas para a Medição:

Provavelmente uma das tarefas mais cruciais para um gerente de projeto seja a elaboração de estimativas de esforço de desenvolvimento, prazo e custo de um projeto de sistemas [32]. Para suportar a realização de tal atividade torna-se necessário a adoção de um método de estimativa de tamanho de sistemas de informação.

A escolha da métrica para auxiliar e suportar as medições de software deve levar em conta alguns parâmetros, como demonstrado a seguir:

- Prover resultados consistentes;
- Permitir sua obtenção por não especialistas em informática;
- Ser de fácil aprendizado;
- Ser compreensível ao usuário final;
- Servir para estimativas;
- Permitir automatização;
- Possibilitar obter séries históricas.
- Técnicas de Estimativas

As técnicas de estimativas podem ser classificadas basicamente, em três categorias:

- Analogia;
- Modelos Algoritmos;
- Análise de Funcionalidade.

1.1. Analogia - Estimativa por Experiência:

Este método, se é que pode ser chamado de método, é baseado na experiência de quem faz estimativas. Quanto mais estimativas feitas, maior é o conhecimento e maior é a possibilidade de acerto.

Como vantagens pode-se citar: baseado em experiência passada, aplicável para projetos com baixo nível de detalhe e o compromisso do grupo que produziu a estimativa.

Como desvantagens pode-se citar: está sujeito à pressão, necessita de *experts* da companhia, pode apresentar grande desvio, é altamente dependente de experiência passada, não deve ser utilizado para projetos grandes e não produz indicadores.

1.2. Modelo Algoritmo – COCOMO:

O método COCOMO (Constructive Cost Model) foi desenvolvido para estimar esforços de desenvolvimento, prazo e tamanho de equipe para um projeto de sistemas. Ele é baseado no número de instruções fontes (número de linhas de código) e supõe que as especificações dos requisitos não serão alteradas substancialmente após a fase de Planejamento e Requisitos.

Como vantagens pode-se citar: baseado em experiência passada, fundamentado em fórmula matemática e pode ser aplicado nas diversas fases do ciclo de desenvolvimento.

Como desvantagens pode-se citar: dependente da tecnologia, dependente de experiência passada e não produz indicadores.

1.3. Análise de Funcionalidade - Análise de Ponto de Função:

O Ponto de Função mede o tamanho do software pela quantificação de suas funcionalidades externas, baseadas no projeto lógico ou a partir do modelo de dados, abrange a funcionalidade específica requerida pelo usuário para o projeto. A funcionalidade requerida diz "o que" será (ou é) entregue para o usuário.

Como vantagens pode-se citar: a estimativa é feita em função da visão do usuário; facilidade de aprendizagem e aplicação da técnica, independência de tecnologia, apoiar e acompanhar a avaliação de produtividade e de qualidade de projetos de software, prover um fator de comparação de softwares, possibilitar a coleta de dados para obtenção de diversos indicadores de acompanhamento, aplicabilidade nas diversas fases de desenvolvimento.

Como desvantagens pode-se citar: necessita acompanhamento constante das medições para gerar os diversos indicadores possíveis, a aplicabilidade nas diversas

fases requer esforço de contagens de pontos de função para cada fase e requer um meio eficiente de armazenamento das informações obtidas nas contagens.

1.4. Comparação entre Métricas:

Cada tipo de medição tem suas vantagens e desvantagens. FPA e COCOMO podem ser utilizadas em conjunto, de modo que se obtenha o melhor de cada uma, conforme as características desejadas nas etapas do desenvolvimento. A efetividade de um método vai depender de quanto as estimativas realizadas se aproximam da realidade. Em complemento é apresentada uma tabela a seguir com os principais pontos necessários em uma métrica [32]:

Estimativas de:	**COCOMO**	**FPA**
Esforço de desenvolvimento	Sim	Sim
Esforço de manutenção	Sim	Sim
Prazo	Sim	Sim
Equipe	Sim	Sim
Esforço de desenvolvimento por fase do projeto	Sim	Sim
Prazo por fase do projeto	Sim	Sim
Equipe por fase do projeto	Sim	Sim
Produtividade	Sim	Sim
Custo	Sim	Sim

Uma consideração deve ser feita com relação à métrica APF é que o padrão definido pelo IFPUG trata somente de medida de tamanho de um sistema de informação em termos de quantidade de pontos de função. A questão da produtividade deve ser tratada particularmente por cada empresa que decide pela implantação desta métrica. Consideração adicional sobre produtividade e qualidade pode ser encontrada no item “14. A Produtividade e a Qualidade Aplicada nas Estimativas:”.

2. Definindo um Programa de Métricas:

A primeira pergunta que deve ser respondida ao apresentar a técnica da análise de ponto de função é: Qual a motivação para a sua utilização?

É fundamental levar em consideração três principais aspectos que devem ser avaliados para respondê-la:

- O contexto de sua aplicação nas organizações que mantêm projetos e operações voltadas à contratação, desenvolvimento e manutenção de sistemas.
- Existe uma análise da problemática presente nesse contexto.
- O entendimento de como a técnica pode ajudar essas organizações a identificar e equacionar o conjunto de questões envolvidas na solução das dificuldades inerentes a esses empreendimentos.

Ao explorar a motivação para medir e os objetos de medição, chega-se à conclusão que o tamanho é umas das propriedades que deve ser medida. Agora é necessário avaliar a melhor unidade para medir o tamanho de sistemas.

A métrica Análise de Pontos de Função é utilizada como geradora de indicadores para estimativas de prazos, gerência de recursos humanos e elaboração de planos de trabalho de projetos, assim como na avaliação e acompanhamento do progresso de projetos e análise da produtividade de equipes. A medida do tamanho de sistemas, em conjunto com tempo e custo, fornece estes indicadores, que constituem um sistema de informações gerenciais - importante ferramenta para a administração da organização. A implantação de um programa de métricas, fundamentado nos indicadores obtidos através da utilização da Análise de Pontos de Função irá possibilitar fornecer uma série de informações, como por exemplo:

Nossa capacidade de responder às solicitações do usuário (produtividade) aumentou 33%. No ano passado, liberávamos 15 Pontos de Função por Homem Mês. Neste ano, estamos liberando 20 PF/ HM.

Nossa capacidade de responder às solicitações do usuário (produtividade) diminuiu 25%. No ano passado, liberávamos 20 PF / HM. Neste ano, estamos liberando 15 PF/ HM. Precisamos fazer algo.

O índice de defeitos foi reduzido de 12% para 10%, resultando num aumento de 16,6% do índice de qualidade.

O custo por ponto de função diminuiu de 100 $ / PF para 80 $ / PF. Este custo menor gerou um aumento de demanda por parte dos usuários.

A precisão de nossas estimativas melhorou significativamente, passando de 45% para 15% de desvios (estimado x atual).

O objetivo principal da aplicação de medições na gestão do projeto está associado aos seguintes aspectos:

- Atingir o prazo inicialmente previsto;
- Atingir o orçamento inicialmente previsto;
- Geração de um produto de software de boa qualidade, adequado ao uso
- Satisfação do cliente / usuário;
- Fornecimento de informações à gerência de desenvolvimento para que possa melhorar, continuamente, os processos de planejamento, desenvolvimento de software e gestão do produto;
- Motivar equipe;
- Direcionar trabalhos - prover informações a tempo de afetar o próprio processo;
- Identificar oportunidades de melhorias - medir impacto de técnicas e ferramentas;
- Prover a gerência de indicadores - Avaliar o ambiente de forma, a saber, se estamos no caminho certo.

Para tanto, é preciso controlar / monitorar o processo de desenvolvimento, visando manter a produtividade nos níveis previstos, remover o quanto antes defeitos introduzidos no produto, reduzindo ou eliminando o esforço de retrabalho, consequentemente mantendo o orçamento sob controle.

O processo de definição de um programa de métricas, isto é, quais métricas e quais informações geradas por estas métricas serão utilizadas pela empresa, deve ser baseado nas necessidades de informação de cada nível organizacional. Isto é obtido a partir do levantamento de informações junto as áreas interessadas. Para o tal, pode-se

aplicar o método Basili GQM (Goal / Question / Metric). O diagrama apresentado a seguir ilustra o modelo Basili.

Goal	Quais são as metas
Question	Qual questão se deseja responder
Metric	Qual Métrica \ Indicador poderá lhe ajudar

3. Dificuldades em se Obter uma Base Histórica de Medições:

Uma vez estabelecido por que medir, a próxima pergunta que deve ser respondida é: O que medir? No caso da área de sistemas devem ser avaliadas não só suas características de produto final, mas também as características dos processos envolvidos em sua concepção e construção. Desta forma primeiro é preciso identificar as características relevantes para a análise.

Para cada um dos objetivos que se deseja acompanhar é possível estabelecer um conjunto de informações que são capazes de a partir do relacionamento entre elas, produzir indicadores para suportar o processo gerencial.

3.1. A Coleta da Base de Dados:

A definição da coleta da base de dados pode ser considerada uma da decisão mais importante, depois que é decidida a implantação da métrica de Análise de Pontos de Função. Esta definição está estreitamente relacionada à quais serão os indicadores desejados.

Como exemplo de informações que podem ser coletadas para compor os indicadores pode-se ter valores estimados e realizados para: homem hora, quantidade de pontos de função, custo, defeitos encontrados, defeitos removidos, além de características do processo de estimativa e fatos ocorridos durante a realização das atividades que estão sendo mapeadas.

A coleta pode ser atomizada até o ponto ideal entre o útil e o desestímulo. Deve ser lembrado que normalmente o trabalho de manter a base de dados atualizada caberá ao responsável pela realização da atividade.

A coleta da base de dados é um dos fatores críticos de sucesso na implantação de um programa de métricas. Corresponde à etapa em que, a partir de um planejamento do que será coletado, feito na fase de definição de um programa de

métrica, define-se como serão armazenadas as informações coletadas (aquisição de uma ferramenta, uso de planilhas, desenvolvimento de um sistema, etc.).

Neste momento, é necessário que todos os envolvidos no processo de coleta estejam motivados e conscientes da importância do trabalho, cujo retorno não será imediato, pois apesar do pequeno acréscimo de trabalho, proveniente das medições e da coleta de dados, ainda não se tem uma massa de dados históricos mínima, tão necessário à uma utilização da métrica.

A atividade de coleta da base de dados tem seu início quando é transmitida a necessidade de um requerimento de negócio, se estenderá ao longo do ciclo de desenvolvimento e acompanhar o sistema de informação durante todo o seu ciclo de vida até sua desativação, quando serão coletados os últimos dados.

Ao final do processo de desenvolvimento devem ser feitas algumas medições, para que dêem subsídios ao aperfeiçoamento continuo dos processos de planejamento de projetos e de desenvolvimento da instalação. Essas medições irão atualizar o banco de dados de métricas e alimentar os indicadores, bem como estabelecer os novos padrões da instalação, os quais serão utilizados para o planejamento e desenvolvimento de novos projetos.

Estas medições compreendem:

- Verificação da exatidão das estimativas;
- Tamanho do software entregue;
- Produtividade do Desenvolvimento;
- Custo do PF de Desenvolvimento;
- Crescimento funcional do software durante o desenvolvimento;
- Reutilização de código;
- Complexidade relativa do software;
- Custo da qualidade;
- Distribuição do esforço por fase;
- Distribuição do custo por fase;
- Densidade e distribuição dos defeitos por fase.

Pode-se ter os seguintes exemplos de indicadores para clientes e para executivos que são resultantes do relacionamento entre as informações coletadas:

- Índice de Qualidade: IQ = Número de Defeitos / Tamanho do Software;
- Tempo médio de solução de problemas;
- Nível de satisfação do cliente;
- Pontos de função liberados;
- Crescimento de software;
- Taxa de produção;
- Qualidade do produto liberado;
- Ranking de defeitos;
- Custo por ponto de função;
- Custo de retrabalho;
- Taxa de faturamento;
- Produtividade;
- Patrimônio de Software e Inventário de Software.

Pode-se ter os seguintes exemplos de indicadores para Gerentes e Equipe que são resultantes do relacionamento entre as informações coletadas:

- Taxa de Produtividade: PF / Pessoa-mês ou PF / Pessoa.hora;
- Produtividade = Tamanho do Software / Esforço;
- Eficiência na remoção de defeitos;
- Estimativa de Esforço: Esforço = Tamanho do Software / Produtividade;
- Retrabalho;
- Taxa de Custo de Software: Taxa de Custo = Custo Total / Tamanho do Software;
- Tendências de custo;
- Custo por ambiente: Custo = Taxa de Custo x Tamanho do Software.

3.2. Medida, Métrica ou Indicador: Qual a Diferença?

A diferenciação destes três termos pode ser melhor entendia quando se descreve cada uma delas segundo o padrão sugerido pelo IEEE [2], [3], [4]:

- Medir / Medida:

  Avaliar, comparando com um padrão. Um padrão ou unidade de mensuração: a extensão, dimensão, capacidade, etc. de alguma coisa, especialmente na forma determinada por um padrão; o ato ou processo de mensurar; o resultado de uma mensuração.

- Mensuração:

  O ato ou processo de mensurar. Um número, extensão ou quantidade obtido através de medição. O ato ou processo de mensurar alguma coisa. Também se refere a um resultado, como um número expressando a extensão ou valor obtido através da medição.

- Métrica:

  Uma medida quantitativa do grau segundo o qual um sistema, componente ou processo possui um dado atributo. Um indicador calculado ou composto, baseado em duas ou mais medidas. Uma medida quantificável do grau segundo o qual um sistema, componente ou processo possui um dado atributo.

- Indicador:

  Um dispositivo ou variável ao qual pode ser atribuído um estado pré-definido, com base nos resultados de um processo, ou na ocorrência de uma condição específica. Como exemplo, um "flag" ou semáforo. Uma métrica que provê uma visão dos processos do desenvolvimento de software e das atividades de melhoria do processo de software com respeito ao alcance dos objetivos.

Um exemplo de medida seria cinco centímetros. O centímetro é o padrão; o cinco identifica quantos múltiplos ou frações do padrão estão sendo considerados. Com o centímetro, uma pessoa medindo alguma coisa nos Estados Unidos vai obter a mesma medida que uma pessoa na Europa.

Um exemplo de uma métrica seria que houve apenas dois erros descobertos pelo usuário nos primeiros 18 meses de operação. Isto dá mais informação significativa do que dizer que o sistema entregue é de excelente qualidade.

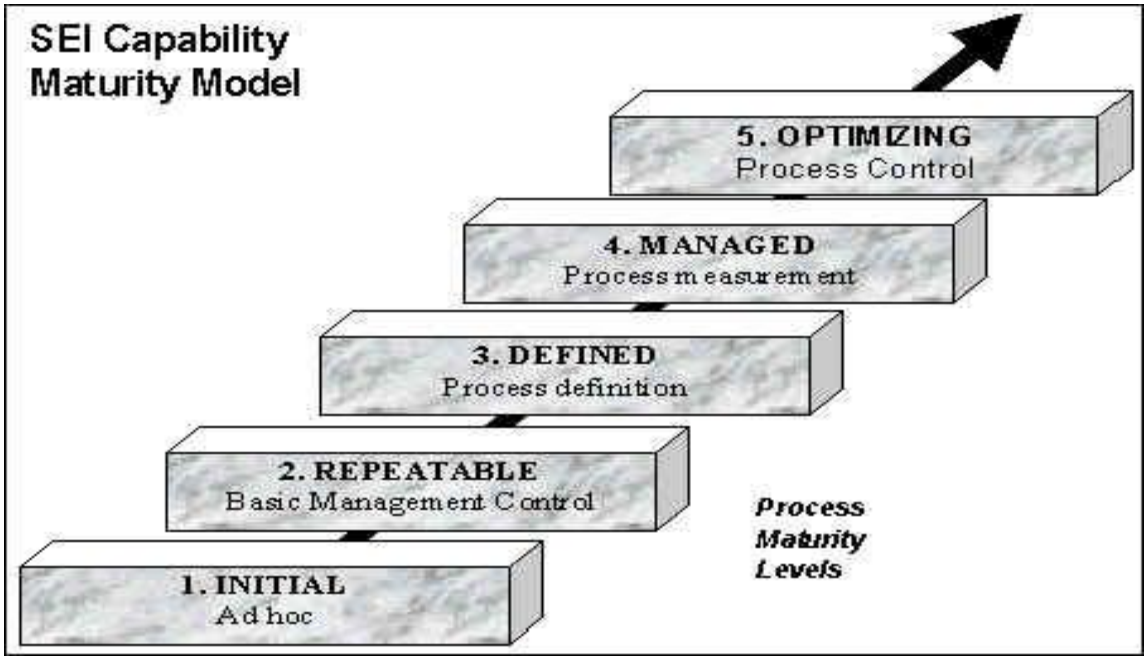
Um exemplo de indicador seria um "flag", um indicador é alguma coisa que chama a atenção de uma pessoa para uma situação específica. Um outro exemplo de

indicador é a ativação de um detetor de fumaça na sua casa; a ele é atribuído um estado pré-estabelecido e um alarme soa se o número de partículas de fumaça no ar excede as condições especificadas para o estado segundo o qual o detetor foi calibrado. Em termos de software, um indicador pode ser um aumento substancial no número de defeitos encontrados na versão mais recente do código.

4. A Questão CMM - Capability Maturity Model:

O CMM é o modelo mais amplamente aceito para o entendimento do processo de desenvolvimento de software. Tem sido utilizado com sucesso por várias organizações na avaliação de seu processo de software, bem como na identificação das áreas-chave onde focalizar as iniciativas de melhoria[7].

O CMM oferece uma estrutura para o entendimento e melhoria da eficácia de uma organização no desenvolvimento de software. É organizado em cinco níveis de maturidade organizacional, onde cada nível representa um estágio evolucionário da capacidade do processo.



Cada nível representa um estágio da maturidade organizacional, descrito em termos de suas "Áreas-Chave de Processos". Uma Área-Chave de Processo (também chamada KPA - *Key Process Area*) é um grupo de atividades relacionadas considerado importante para que uma organização funcione no nível apropriado de maturidade do processo.

Os objetivos de uma Área-Chave de Processo são alcançados quando certas atividades são executadas. Por exemplo, um dos objetivos da Área-Chave de Processo do Nível 2, *Planejamento do Projeto de Software*, é "Estimativas de software são

documentadas para utilização no planejamento e supervisão do projeto de software."
A tabela a seguir, apresenta as Áreas-Chave de Processo para cada Nível do CMM.

Nível – 5 **"Otimizado"**	Prevenção de Defeitos Gerenciamento da Mudança Tecnológica Gerenciamento da Mudança no Processo
Nível - 4 **Gerenciado"**	Gerenciamento Quantitativo do Processo Gerenciamento da Qualidade de Software
Nível – 3 **"Definido"**	Foco no Processo Organizacional Definição do Processo Organizacional Programa de Treinamento Gerenciamento Integrado de Software Engenharia do Produto de Software Coordenação Intergrupos Revisões de Parceiros
Nível – 2 **"Repetível"**	Gerenciamento de Requisitos - RM Planejamento do Projeto de Software – PP Acompanhamento e Supervisão do Projeto de Software - PMC Gerência de Subcontratação de Software – SAM Medição e Análise - MA Garantia da Qualidade de Software – PPQA Gerenciamento de Configuração de Software – CM
Nível – 1 **"Inicial"**	(nenhuma Área-Chave de Processo)

Observa-se que o Nível 1 não possui Áreas-Chave de Processo. Na verdade, o nível "Inicial" representa a ausência de processos. Diz-se que uma organização está funcionando no Nível 2 quando *todos* os objetivos das Áreas-Chaves de Processo do Nível 2 foram alcançados. A maioria das organizações ainda está buscando o Nível 2.

A principal preocupação do Nível 2, o nível "Repetível", é a boa prática do gerenciamento de projetos. Este nível focaliza os processos aplicáveis ao nível do projeto.

O Nível 3, por outro lado, preocupa-se com processos relacionados às atividades de engenharia no âmbito da organização.

O Nível 4 olha os processos com uma granularidade mais fina, oferecendo mensuração e feedback ao nível do processo.

O Nível 5, alcançado por uma pequena quantidade de organizações, focaliza a melhoria contínua de uma organização de classe mundial.

Devido ao foco do Nível 2 no gerenciamento de projetos, é nas Áreas-Chave de Processo desse nível que é iniciada a aplicação da Análise de Pontos de Função.

As atividades e objetivos relevantes para cada Área-Chave de Processo do Nível 2 (vide tabela anterior) estão em conformidade com o documento CMU/SEI-93-TR-25 do SEI:

4.1. Gerenciamento de Requisitos:

Os requisitos de sistema alocados ao software são controlados, de modo a estabelecer uma baseline para utilização da gerência e da engenharia de software.

O grupo de engenharia de software revisa os requisitos alocados, antes que os mesmos sejam incorporados ao projeto de software.

As mudanças nos requisitos alocados são revisadas e incorporadas ao projeto de software.

A Análise de Pontos de Função pode ser utilizada para descrever e documentar requisitos funcionais. Este objetivo é suportado pela análise dos requisitos, com a finalidade de quantificar o tamanho funcional do projeto. Uma contagem de pontos de

função na fase de requisitos de um projeto pode ser utilizada em estimativas e para fixar uma baseline para gerenciar o aumento do escopo.

Conforme os requisitos mudarem, os pontos de função podem ser usados para comunicar o tamanho das mudanças relativas ao tamanho do projeto.

4.2. Planejamento do Projeto de Software:

Estimativas de software são documentadas para utilização no planejamento e supervisão do projeto de software.

O plano do projeto de software é documentado. Estimativas de tamanho dos produtos de software e quaisquer mudanças nos produtos de software são documentadas.

Estimativas do tamanho dos produtos de software (ou de mudanças no tamanho dos produtos de software) são derivadas de acordo com um procedimento documentado.

Estimativas de tamanho são obtidas para todos os produtos e atividades relevantes. Um exemplo de medida de tamanho de software são os pontos de função.

Estimativas para o custo e esforço do projeto são derivadas de acordo com um procedimento documentado.

4.3. Acompanhamento e Supervisão do Projeto de Software

Resultados e desempenhos reais são comparados com os planos de software.

Os tamanhos dos produtos de software (ou o tamanho das mudanças nos produtos) são acompanhados e são tomadas as ações corretivas necessárias.

Dados correspondentes a medidas reais e de re-planejamento do projeto de software são registrados.

Nesta Área-Chave de Processo, as contagens de pontos de função são atualizadas ao longo da vida do projeto. Registrando, acompanhando e analisando as mudanças de tamanho ao longo do projeto, uma organização pode melhorar a eficácia

e a acurácia de suas estimativas, melhorando dessa forma o controle e supervisão do projeto.

4.4. Gerenciamento de Subcontratação de Software:

O contratado principal acompanha os resultados reais e desempenho do subcontratado, em relação aos compromissos assumidos.

O trabalho a ser subcontratado é definido e planejado, de acordo com um procedimento documentado.

A Análise de Pontos de Função tem sido amplamente utilizada em situações de gerenciamento de subcontratos, com a finalidade de comunicar o tamanho, avaliar propostas e definir termos contratuais. Isto inclui tanto desenvolvimentos isolados, quanto contratos de terceirização (outsourcing). Esta Área-Chave de Processo sugere a utilização de pontos de função para estimativas na fase de requisitos e acompanhamento ao longo do projeto. Uma contagem final na conclusão do projeto é utilizada para comparar o que foi entregue com o que foi planejado, sendo útil para ajudar a avaliar o desempenho do contratado.

4.5. Gerenciamento de Configuração de Software:

Os produtos de software selecionados são identificados, controlados e disponibilizados.

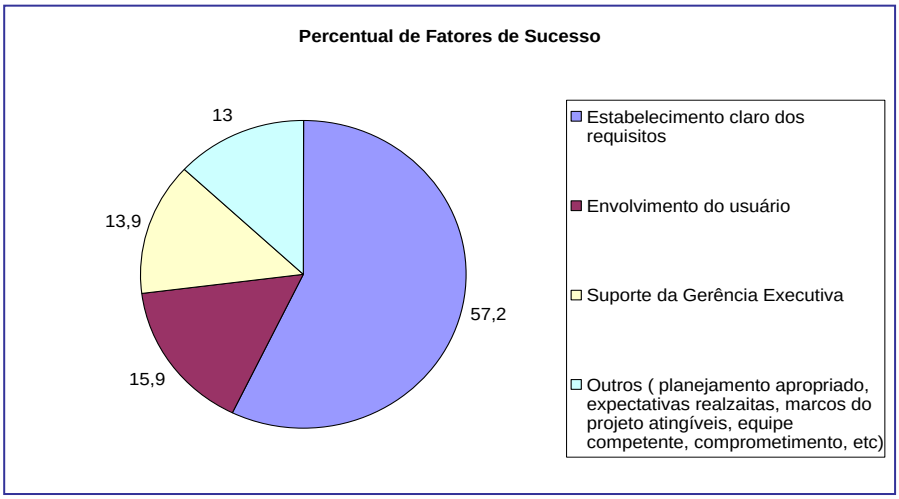
As mudanças nos produtos de software identificados são controladas.

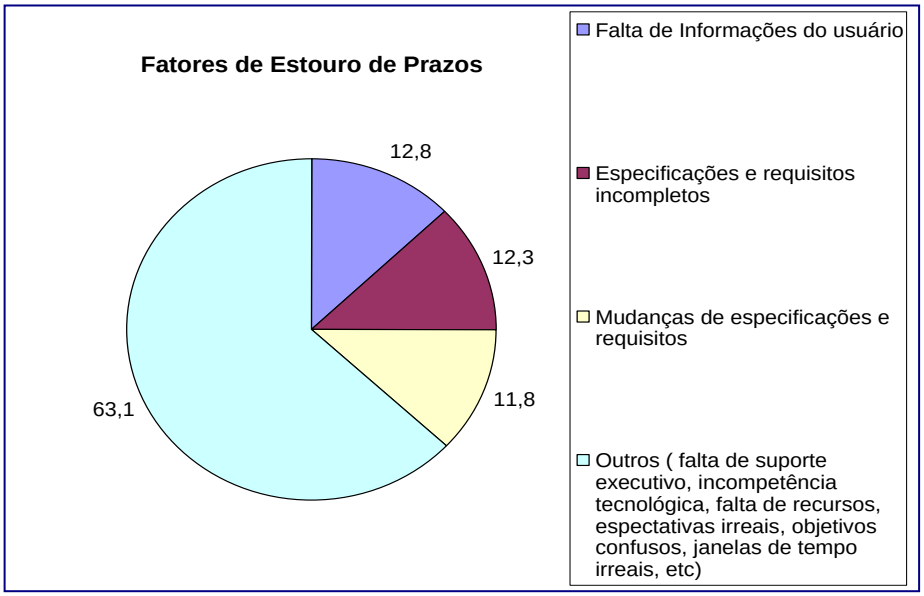
As mudanças nas baselines são controladas de acordo com um procedimento documentado.

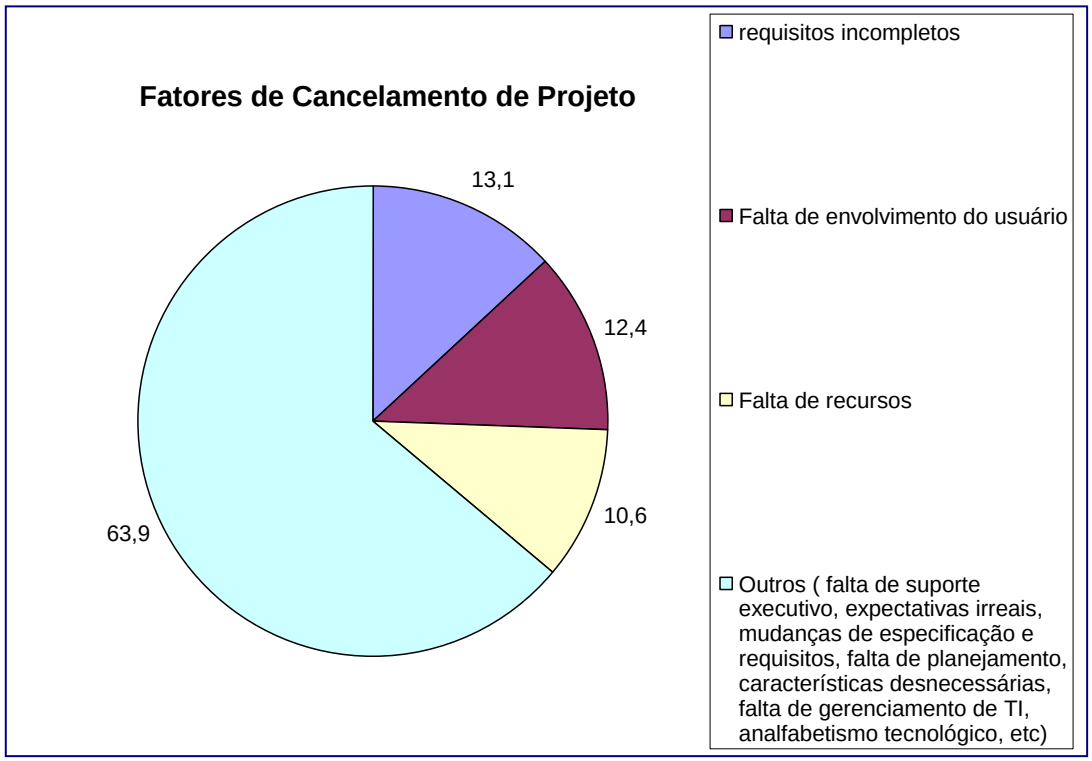
Os Pontos de Função desempenham um papel importante no suporte ao Gerenciamento de Configuração de Software. Quando os requisitos mudam, o tamanho das mudanças deveria ser quantificado e expresso em pontos de função. A documentação dos pontos de função do projeto também pode ser colocada sob o gerenciamento de configuração, constituindo uma técnica para descrever as mudanças funcionais.

4.6. A importância da Gestão de Requisitos:

A seguir serão apresentados três gráficos que demonstram o motivo de se considerar a gestão de requisitos como sendo um dos principais fatores de sucesso no desenvolvimento de sistemas de informação [30] [Fonte: CHAOS Reports – The Standish Group].







Com o intuito de diminuir os problemas relacionados ao pouco envolvimento dos usuários e à deficiência dos processos de definição e gerenciamento de requisitos, causadores dos impactos negativos que levam ao atraso e mesmo ao cancelamento de projetos de software, torna-se necessário aumentar a proximidade entre os usuários e a equipe responsável pelo sistema. Diminuir esta distância significa:

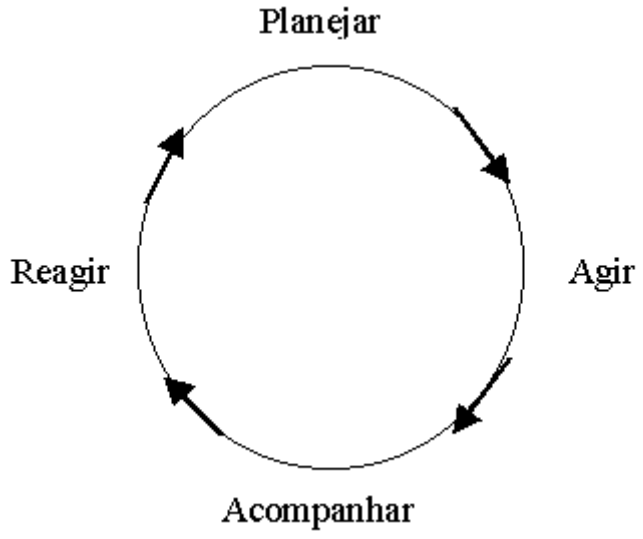
- Aumentar a interatividade entre todos aqueles que estão direta ou indiretamente interessados no projeto, por intermédio de um processo de trabalho que preveja a maturidade sucessiva dos requisitos ao longo de cada fase do desenvolvimento;
- Aproximar as visões dos usuários e dos desenvolvedores com relação aos requisitos a serem implementados no sistema.

5. Melhoria do Processo de Software Através de Indicadores da Qualidade e Produtividade:

Apesar dos grandes benefícios que podem ser obtidos com a utilização de um programa de métricas para o gerenciamento dos projetos e produtos de software, o fator humano é um item extremamente sensível que deve ser levado em consideração e que, se negligenciado, pode conduzir ao fracasso tentativas bem intencionadas de medições.

O princípio básico que norteia as regras, que deve orientar o comportamento ético da gerência e das equipes de desenvolvimento, fundamenta-se no que podemos considerar um axioma enunciado por Edward Deming (*Out of the Crisis*, MIT Press, 1986): "85% dos problemas que ocorrem nos processos da empresa são de responsabilidade da gerência". Consequentemente, jamais se devem avaliar desempenhos individuais e sim enfocar processos e produtos.

Um método da qualidade utilizado para implementação de melhoria contínua no processo é o Ciclo PDCA. O P (Planejar) é o Planejamento da mudança no processo, o D (Do - Agir) é a execução da mudança planejada, o C (Check - Acompanhar) é a verificação dos resultados obtidos pela ação executada e o A (Act - reagir) é a decisão de mudar, baseada nos dados analisados.



Em cada uma destas atividades são definidas sub-atividades que e produtos que as compõem:

Planejar:

- Metas
- Objetivos
- Necessidade de Informações
- Fatores Críticos de Sucesso
- Problemas e Oportunidades

Agir:

- Criação de medidas, padrões e coleta de dados

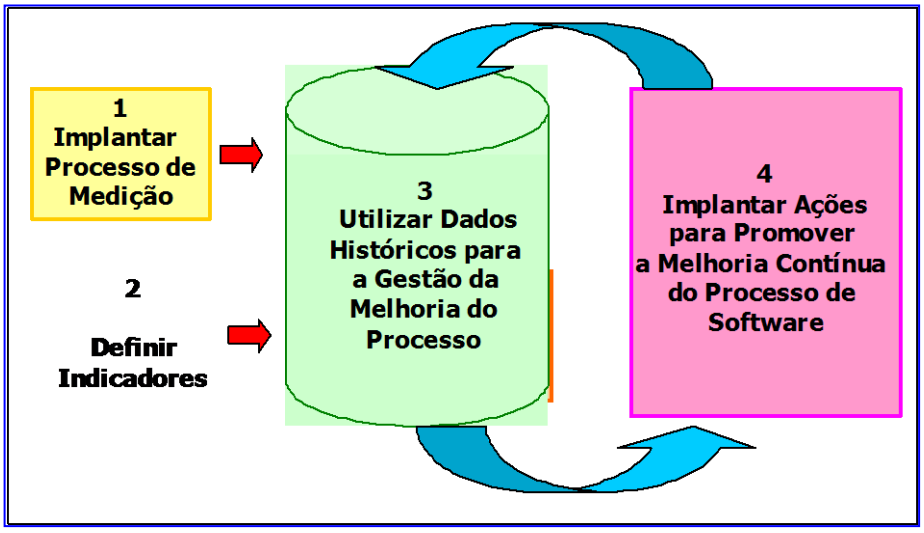
Acompanhar:

- Acompanhar o andamento físico e financeiro dos projetos e serviços
- Verificação dos projetos com desvio de estimativas de prazos
- Verificação dos projetos com desvio de estimativas de custos
- Análise de Impacto (avaliar as variações de qualidade, produtividade e aspectos de custos dos processos de desenvolvimento e gestão do software, em função de mudanças no ambiente)
- Análise de Tendências (comportamento futuro dos principais atributos do processo de planejamento de projetos, desenvolvimento de software e gestão de produto)
- Comparação entre plataformas e tecnologias

Reagir:

- Incentivar / Premiar
- Treinar
- Alocar / Realocar Recursos
- Modelar o Ambiente de Desenvolvimento
- Eliminar os problemas antes deles te eliminarem

A implantação de uma Metodologia que pode ser graficamente apresentada como a figura a seguir, possui quatro fases que se interagem continuamente. Isto é, Após a decisão, definição e Implantação de um processo de medição de software, o processo de definição dos indicadores é realizado de modo a atender as necessidades atuais de negócio. Porem, esta necessidade não pode ser vista como uma atividade que é feita uma única vez. As necessidade evoluindo e novos indicadores precisão ser criados. A Utilização de dados históricos para o planejamento e controle do projetos é fundamental e a base é alimentada continuamente com os dados que vão sendo extraídos dos projetos em andamento e dos finalizados.



A partir da Base Histórica são extraídas informações baseadas nos indicadores para suportar o processo gerencial de melhoria contínua. Portanto, os principais objetivos da utilização dos indicadores propostos são:

- Melhorar a qualidade do processo de desenvolvimento de software;
- Aumentar a produtividade do processo de desenvolvimento de software;
- Melhorar a qualidade do software desenvolvido;
- Auxiliar gerentes a planejar e a controlar os projetos de software;
- Promover a melhoria contínua do processo de software.

Para implantar a melhoria contínua no processo de desenvolvimento de software, é indispensável uma gerência eficaz dos parâmetros quantitativos do processo que resultam em informações de controle [Metodologia para Melhoria do Processo - Claudia Hazan (5)]. Portanto, para manter-se um processo de software sob controle deve-se instituir um processo de medição eficaz, visando a coleta de dados para os indicadores, os quais serão utilizados para manter o processo sob controle.

Uma organização deve identificar quais são as maiores áreas de riscos e focalizar as oportunidades de melhoria para corrigir ou minimizar os fatores de risco, dando prioridade às melhorias com base nos seguintes fatores estratégicos de sucesso, críticos para a organização na perspectiva do negócio: [JONES, 1997].

- Redução do tempo de entrega para o mercado;
- Redução dos custos do desenvolvimento e manutenção do software;
- Melhoria na precisão das estimativas de custos e cronogramas;
- Uso otimizado de novas tecnologias;
- Uso eficiente de consultores e contratos de terceirização.

A coleta de dados históricos de desempenho (cronogramas, custo, qualidade e produtividade) deve ser uma das primeiras ações dentro do processo de melhoria contínua com a finalidade de obter-se uma base de comparação. Utilizam-se os dados do processo tanto para analisá-lo quanto para modificá-lo no sentido de prevenir problemas e melhorar a eficiência.

Esses dados históricos darão subsídios para a criação de indicadores de desempenho financeiro, qualidade e produtividade que retratem o desenvolvimento de sistemas de informação No esforço de implantação de uma nova mentalidade voltada para a qualidade e produtividade, não é admissível o desenvolvimento de sistemas através de ‘feeling’, onde questões como: qual é a produtividade da área de informática, qual é a capacidade de produção, qual conjunto de ferramentas possibilita a maior produtividade, quais são os indicadores de qualidade existentes, qual caminho devo adotar, desenvolver ou comprar um pacote e customizá-lo, são simplesmente deixadas de lado ou respondidas sem suporte de uma base quantificável.

Medições operacionais devem ser utilizadas a fim de permitir: a análise de tendências de determinados indicadores, que podem subsidiar ações para reversão ou sustentação dessas tendências; a análise de impactos na introdução de novas tecnologias sobre a qualidade e produtividade, que pode auxiliar na decisão sobre quais combinações de elementos de tecnologia garantem melhores resultados; a análise de atributos, que permite a comparação da qualidade e produtividade entre plataformas, metodologias, áreas de aplicação, habilidades técnicas de pessoas e assim sucessivamente.

6. O Choque Cultural Envolvendo Estimativas:

Nas fases iniciais do projeto, como por exemplo, durante o levantamento de requisitos, ainda não há o conhecimento completo das características do produto que permita a apuração de sua futura dimensão. Nesse caso é necessário estimar. Vários modelos de estimativas foram criados para fornecer métricas que permitam atender com menor margem de erro às necessidades de comunicação e informação do projeto.

A análise de pontos de função permite não só medir o tamanho do sistema em termos de funcionalidades fornecidas ao usuário, mas também estimar seu tamanho em qualquer fase do seu ciclo de vida (mesmo que os requisitos ainda não tenham sido detalhados). A manutenção de registros de outros projetos semelhantes, com a evolução das estimativas iniciais até a medição final, permite um acompanhamento da relação entre a quantidade de pontos de função estimados ou calculados nos vários estágios de conhecimento do produto.

As medidas obtidas nos diversos momentos do projeto não só têm valor quando avaliadas isoladamente, mas também quando relacionadas entre si. Ao comparar a evolução do tamanho funcional durante a evolução de projetos passados, pode ser obtido um fator que indique o percentual de crescimento funcional entre essas diversas etapas. Ao identificar projetos passados com características semelhantes ao que se está planejando, é possível ajustar as estimativas realizadas em fases iniciais.

Contudo, tamanho em si tem pouco valor analisado de forma isolada. Ele passa a ser relevante quando serve de base para responder às perguntas referentes a outros aspectos. O planejamento de projetos de sistemas envolve equacionar um conjunto complexo de elementos, entre eles:

- Quais produtos devem ser desenvolvidos;
- Por meio de que atividades;
- Envolvendo quanto de esforço;
- Quais profissionais serão alocadas;
- Durante quanto tempo;
- Quais riscos devem ser identificados e contingênciados.

Dentre os diversos paradigmas que a implantação de um programa de métrica necessita quebrar está a questão da estimativa de custo. O cálculo dos custos de um projeto requer o somatório de um conjunto de custos das várias atividades que compõem o processo de desenvolvimento de sistemas de informação. Portanto, a atividade de padronização do cálculo de custos de projeto de software requer uma padronização prévia de processo desenvolvimento de software. Além, disto é necessário acreditar em um padrão que ainda tem uma margem de desconhecimento por não ainda amadurecimento dentro da empresa.

6.1. A Estimativa de Custo:

A maioria das organizações calcula o custo do software a menor, por larga margem. O verdadeiro custo do software é a soma de todos os custos durante a vida do projeto, incluindo-se aí todos os custos esperados de melhoria e manutenção. De fato, o cálculo real seria o valor presente de todos os desenvolvimentos, melhorias e manutenções no decorrer da vida do projeto. Este tipo de análise demonstra a recompensa de se investir na fase inicial da análise e projeto.

Quanto mais se investir nas fases iniciais, mais será economizado na redução dos custos de manutenção e melhoria. É importante ser capaz de calcular o custo unitário, a fim de avaliar o investimento inicial e compará-lo aos gastos futuros. O custo unitário pode ser em horas / PF ou $ / PF. Os aumentos no investimento inicial deveriam reduzir proporcionalmente os custos unitários das futuras melhorias e manutenções.

A utilização de Pontos de Função para auxiliar na estimativa de custo, elaboração de cronograma e esforço do projeto é uma questão de associação entre as seguintes informações necessárias ao controle de projetos de software:

- Quantidade de pontos de função da atividade;
- Estrutura metodológica adotada pela empresa (ciclo de desenvolvimento);
- Índice de Produtividade para a realização da atividade;
- Valor de taxa hora relativo ao perfil do profissional que executará a atividade;

A Combinação destas informações possibilita fornecer os números necessários para a elaboração do cronograma do projeto.

6.2. Prazos e Custos Dados pelos Patrocinadores e Alta Gerência antes Mesmo de ser Definido o Problema:

A implantação de um processo de estimativa deve ser associado a uma mudança cultural do processo de orçamento de desenvolvimento de software. Não é possível dar preços a um produto sem antes ter uma idéia de seu escopo e tamanho.

A alta gerencia deve estar consciente que para se ter uma idéia de custo, um trabalho mínimo deve ser realizado para o mapeamento do problema e assim, se reunir um conjunto de informações que subsidie os cálculos de custos e prazos.

6.3. Como Fazer Mudanças:

A implantação de um programa de métrica deve ser tratado do mesmo modo da implantação de um sistema de informação. Todo o ambiente de trabalho será afetado de uma forma ou de outra e uma nova mentalidade deverá ser transmitida e assimilada por todos os envolvidos.

Deve ser criado um compromisso entre todos os envolvidos. O método a ser implantado deve ser sistemático e disciplinado para proporcionar transparência e agilidade no processo de mudança organizacional, reduzindo custos e maximizando os resultados.

Para que a mudança aconteça, são necessários alguns requisitos e condições. Dentro de um contexto organizacional, a mudança acontecerá na medida em que 3 fatores aconteçam.

Causa	Solução	Implementação
Existe real Necessidade da Mudança	Há uma Alternativa Coerente	Condução do Projeto de Mudança

É na fase de implementação que muitos Projetos de Mudança fracassam, apesar de ser reconhecido a "causa" e de ter identificado a melhor "solução", a falta de metodologia adequada de "implementação" conduz ao fracasso de não atingir os resultados esperados. Portanto, algumas atividades podem ser planejadas e executadas de modo a minimizar os riscos de fracasso na implantação:

- Elaborar um diagnóstico claro da situação:
  Levantar dados e informações com o maior número de pessoas envolvidas no processo de mudança. Entender o Projeto e seus objetivos.
- Alinhamento dos Objetivos:
  A definição do escopo, especificação funcional e o resultado final desejado, devendo ser quantificado na medida do possível.
- Fixar Estratégia:
  Estabelecer a metodologia geral ou plano básico para se implementar a mudança. Definir a política de utilização de recursos internos e externos.
- Fortalecimento da Cultura Organizacional:
  O desafio consiste na necessidade de construir uma nova cultura e/ou reforçar a cultura existente.
- Comprometimento das Pessoas com os Objetivos e Estratégia:
  O sucesso de tais mudanças depende da capacitação e preparação do pessoal, em todos os níveis, para enfrentar os novos desafios. É preciso ousar, arriscar e inovar para vencer. A preparação do pessoal para mudança exige uma abordagem inovadora que demonstre, através de vivência, como encarar o risco e desenvolver formas inovadoras de trabalho.
- Identificar Atividades:
  Subdividir o projeto como um todo em "fases do trabalho" , passiveis de serem gerenciadas e controladas. A "estrutura analítica" deve ser aplicada ao projeto de mudança.
- Identificar Recursos:

Determinar os Recursos humanos disponíveis para implementar as mudanças, analisando a qualificação, tempo de utilização e quantidade requerida. Levantar também necessidades de recursos financeiros, materiais e equipamentos especiais.

- Treinamento Adequado das Pessoas:

  É necessário dotar os programas de treinamento de conteúdo e metodologia, embasados em uma filosofia que devolva ao indivíduo a capacidade de pensar, a partir dos resultados da sua própria experiência. Isso implica num processo de aprendizagem que é desenvolvido além da sala de aula, integrando técnicas e pensamento gerencial, intuição e intelecto, ciência e bom senso.

- Estabelecer Tempo para cada Atividade:

  Com base nos recursos, calcular o tempo de execução de cada atividade. Estabelecer calendário das atividades com datas programadas de início e término.

- Rever Tudo:

  Ajustar as necessidades de prazo global com a disponibilidade de recursos, reavaliar a lógica, refinar o plano. Em todas as fases, o gerenciamento eficaz dos aspectos culturais e humanos é a chave para o sucesso. O desafio é conseguir integrar a Abordagem Técnica da Implementação com o Desenvolvimento das Competências Humanas.

7. Influência da Metodologia de Desenvolvimento e Gestão de Projetos:

7.1. A Questão da Metodologia de Desenvolvimento Rápido:

Os profissionais de TI de hoje estão enfrentando um dilema: por um lado sabem, a partir de penosa experiência, que desenvolver sistemas de informação complexos com alta qualidade utilizando uma perspectiva informal, de *hacker*, é arriscado. Mas também sabem que as abordagens de engenharia formais e disciplinadas, normalmente associadas à ISO-9000 e ao SEI-CMM, são tão burocráticas e consomem tanto tempo que tornam impossível o cumprimento dos prazos cada vez mais agressivos e competitivos do ambiente e do chamado "tempo Internet".

Podemos caracterizar as metodologias formais e disciplinadas da engenharia de software como "pesadas" – não apenas em termos do peso da documentação em papel que produzem, mas também em relação ao grau de esforço gerencial, revisões de QA e procedimentos rígidos que os desenvolvedores devem seguir. Por outro lado, as abordagens para o desenvolvimento tais como o "RAD" (desenvolvimento rápido de aplicações) e a prototipação poderiam ser caracterizadas como "leves" – não apenas porque tendem a produzir uma quantidade mínima de documentação em papel, mas também porque minimizam o grau de esforço gerencial. Infelizmente, muitos dos projetos RAD nos anos 90 foram tão "leves" em sua abordagem metodológica que os mesmos foram quase inexistentes; em retrospectiva, tais projetos freqüentemente degeneraram para exercícios de *hacking*, sem praticamente qualquer documentação.

Certamente é necessário um equilíbrio entre os extremos de não ter nenhuma metodologia e o de ter um excesso de metodologia. Talvez a metodologia "equilibrada" mais popular hoje em dia seja "XP", conforme explicada por Kent Beck em seu livro "*eXtreme Programming eXplained*" ("Programação eXtrema explicada"). Uma outra é a metodologia SCRUM criada por Ken Schwaber. A maioria dos veteranos de software está familiarizada com a filosofia dessas metodologias "leves"; com efeito, Peter DeGrace e Leslie Hulet Stahl as resumiram há uma década, em

"*Wicked Problems, Righteous Solutions: A Catalog of Modern Software Engineering Paradigms*" (Prentice-Hall, 1990) ("Problemas Perversos, Soluções Corretas: Um Catálogo de Paradigmas Modernos de Engenharia de Software").

As metodologias leves representam uma abordagem consciente para o investimento de tempo, dinheiro e recursos nas diversas atividades associadas ao desenvolvimento. Porerm, quanto pode ser considerado excesso de análise de requisitos, e quanto devem ser considerados insuficientes? Uma abordagem "leve" para os requisitos poderia consistir de documentar cada um dos requisitos (normalmente centenas) associados a um projeto de desenvolvimento com uma única frase sucinta. Uma abordagem "média" seria documentar cada um com um parágrafo de texto. Uma abordagem "pesada" exigiria modelos UML detalhados, definições de elementos de dado e descrições formais dos "métodos" associados a cada objeto.

A escolha entre uma abordagem leve ou pesada para os requisitos provavelmente sofrerá forte influência da pressão corporativa relativa ao "*time to market*" (tempo para a entrega do produto). Da mesma forma, a taxa de rotatividade dos empregados é um fator: uma das justificativas para a existência de um processo formal de desenvolvimento de software é que um documento detalhado descrevendo os requisitos, o *design* e o código irá diminuir o caos se os principais desenvolvedores saírem no meio de um projeto. Ao mesmo tempo, se por um lado os sistemas dos anos 70 e 80 tinham uma vida útil esperada de uma ou duas décadas, talvez uma empresa "ponto-com" esteja disposta a assumir um compromisso formal de que suas aplicações de *e-business* existirão durante apenas um ano, para depois serem completamente reescritas. Se for esse realmente o caso, e se a aplicação da próxima geração for completamente diferente da atual, será que fará sentido seguir uma abordagem pesada, apenas porque é um pré-requisito para alcançar o nível 3 do SEI-CMM?

Igualmente, quanto formalismo e rigor são apropriados para as fases de *design* e teste? Que quantidade de documentação é apropriada, com relação ao preenchimento de folhas de tempo (*time sheet*), relatórios de progresso, atas de reunião e outras atividades normalmente associadas ao gerenciamento de um projeto – especialmente para projetos que duram apenas uma semana? Essas perguntas sempre

foram relevantes, mas as respostas que sempre aceitamos como estratégias corporativas precisam ser reexaminadas de tempo em tempo, pois os parâmetros de custo/benefício mudam, assim como a tecnologia e os desenvolvedores de software.

As metodologias leves também reexaminam as suposições que temos feito, historicamente, a respeito de investir recursos em análise de requisitos e na melhoria de processos. Em 1981, o livro "*Software Engineering Economics*" ("A Economia da Engenharia de Software") de Barry Boehm forneceu-nos a surpreendente revelação de que, se um erro fosse cometido na fase de análise de um projeto, seria 10 vezes mais barato identificá-lo durante a essa fase, do que permitir que ele fosse detetado somente na fase de *design*. Porém, Boehm fez uma suposição que pode não ser verdadeira na primeira década do novo milênio: o crescimento dos números é relevante somente se for possível identificar o defeito na fase do ciclo de vida na qual ele ocorrer. No ambiente de hoje, há muitas situações nas quais isso não é possível. Quando Bill Gates sentou-se para formular os requisitos para o *Internet Explorer*, talvez ele tenha dito, "Faça-o exatamente como o *Netscape Navigator*, porém ainda melhor". E, talvez, Marc Andreesen da *Netscape* tenha pensado: "Farei o *Navigator* como o *Mosaic*, porém ainda melhor". Mas o que pensou Tim Berners ao criar a versão inicial da *World Wide Web* e a primeira versão, grosseira, de um *browser*? O que adiantaria para ele escrever requisitos detalhados?

De maneira similar, os defensores das metodologias pesadas dizem que, se um defeito for encontrado durante o desenvolvimento, a culpa não deveria ser atribuída ao indivíduo que o criou; ao invés disso, tal fato deveria conduzir ao exame do processo que permitiu que o defeito acontecesse. Porém, mais uma vez, existe uma suposição fundamental – isto é, a única razão pela qual vale a pena investir recursos para identificar a falha em um processo é porque pretendemos utilizá-lo novamente – porque nosso próximo projeto será suficientemente parecido com o anterior, de forma que poderemos usar o mesmo processo. Agora, as coisas estão mudando tão rapidamente, que não há garantia de que o projeto N+1 terá qualquer semelhança com o projeto N; assim sendo, o processo de ontem pode ter de ser substancialmente modificado para ser usado amanhã. Então, talvez não valha a pena realizar o esforço

para consertar nada que não seja uma grande falha no processo, pois os respectivos detalhes serão úteis apenas para um único projeto.

Naturalmente, há circunstâncias sob as quais poderemos continuar a depender dos antigos e fundamentais princípios da engenharia de software e que podem justificar uma abordagem metodológica pesada. Devemos nos perguntar, todavia, se as suposições subjacentes a esses princípios ainda são válidas. Para muitos dos projetos de hoje, as suposições subjacentes precisam ser modificadas, e as metodologias leves constituem a melhor abordagem em termos de eficácia e custos.

É importante destacar que o processo de estimativa requer padronização, requer índices de produtividade aplicados a estes padrões. A metodologia de desenvolvimento seja ela para desenvolvimento rápido, seja para outro tipo de desenvolvimento deve ser padronizada e as informações para serem armazenadas na base histórica de estimativas devem ser compatíveis com os métodos de desenvolvimento adotados pela empresa.

7.2. Influência do Ciclo de Vida na Estimativa:

O ambiente atual de desenvolvimento de software é uma miríade de metodologias de desenvolvimento, plataformas, configurações e abordagens centralizadas e descentralizadas. Independentemente da metodologia e do ambiente em questão, contudo, o pré-requisito principal para a qualidade no desenvolvimento de sistemas é a documentação e o entendimento dos requisitos do sistema. Peter Morris, em seu livro sobre a história da gerência de projetos, concluiu que "os requisitos estavam no topo da lista das causas da falta de controle dos projetos".[2]

Concentrando-se esforços para descrever os requisitos dos softwares com casos de uso e medindo estes requisitos com Pontos de Função, os projetos podem ser controlados e gerenciados de maneira muito mais ampla do que são atualmente. O aumento do foco nos requisitos leva a testes mais efetivos dos produtos e resulta em software de melhor qualidade, assim como reforça muito dos princípios resumidos por Watts Humphrey no novo Team Software Process (TSP) [3].

7.3. Requisitos de Software:

Nem todos os requisitos do software são especificados ou conhecidos no mesmo nível ao longo de um projeto. Desta forma, eles se enquadram em uma das três categorias mostradas adiante:

- Conhecidos:

  Requisitos conhecidos, básicos, essenciais, que definem as funcionalidades do negócio que serão suportadas ou executadas pelo software. Geralmente bem articulados e documentados, tanto nos projetos de substituição quanto nas melhorias de sistemas. Estes requisitos representam a funcionalidade do sistema existente que será também requerida no novo sistema.

- Desconhecidos, Previsíveis:

  Estes requisitos, assim como os Conhecidos, são exigências mandatórias no software. Embora nos estágios iniciais do projeto estes requisitos sejam desconhecidos, a equipe de projeto sabe que estes requisitos tornar-se-ão conhecidos durante o desenvolvimento. Freqüentemente os requisitos deste tipo são rotulados pelo termo em inglês "scope creep", (crescimento descontrolado do escopo), e podem montar a algo entre 25% e 50% do tamanho final do software desenvolvido[3]. Exemplos que normalmente caem dentro desta categoria são relatórios, ou consultas a dados que são previstas, mas não completamente definidas. Esta área dos requisitos é gerenciável através do planejamento de risco ou de contingência. Adicionalmente, os dados históricos são freqüentemente uma ferramenta para prever-se a percentagem de scope creep que pode ocorrer.

- Desconhecidos, Imprevisíveis:

  Este é o mais imprevisível tipo de requisito, porque nem mesmo o histórico pode fornecer informações adequadas para acomodar o Desconhecido, Imprevisível. Estes requisitos freqüentemente surgem tarde no ciclo de desenvolvimento, e a gerência formal de mudanças é crítica para controlar o seu impacto. A categoria Desconhecida / Imprevisível inclui alterações nos requisitos existentes, bem como

novas funcionalidades que foram negligenciadas ou que surgiram após a fase de definição de requisitos. Este tipo de requisito tem o potencial de influenciar no crescimento da espiral de re-trabalho do projeto.

No momento que o projeto atinge a etapa de teste, a percentagem relativa de requisitos conhecidos e desconhecidos cresce dramaticamente. O desafio das equipes de desenvolvimento é maximizar o número e a clareza dos requisitos conhecidos nas etapas iniciais do projeto, bem como gerenciar e controlar o impacto dos requisitos desconhecidos quando surgirem.

Ferramentas e metodologias para formalizar os requisitos podem ajudar na maximização do conhecimento nas etapas iniciais do projeto, permitindo uma estimativa mais acurada do projeto. Usando uma analogia com a construção de uma casa, quando mais detalhada e completa for a planta baixa, mais acuradas poderão ser as estimativas do projeto. Quando um projeto gasta o tempo necessário para certificar-se de que seus requisitos estão bem definidos e completos (ambos Conhecidos e Desconhecidos, *Imprevisíveis*), é o mesmo que garantir que a planta baixa esteja completa antes que o modelo final seja desenhado e que as fundações da casa sejam iniciadas. Os softwares de alta qualidade surgem da maximização do número de requisitos conhecidos nas etapas iniciais do projeto e da gerência do impacto dos requisitos desconhecidos quando finalmente tornam-se conhecidos.

8. Utilização de Casos de Uso para Determinar o Número de Pontos de Função:

Um método para documentação dos requisitos do usuário, utilizado na análise orientada a objetos, é chamado "casos de uso" (use cases) e pode ser aplicado à maioria dos projetos de desenvolvimento. Um Caso de Uso resume, em termos do usuário de negócio, uma "unidade de trabalho" lógica e completa que a aplicação pode suportar, sem referência a como a função será implementada. Os casos de uso descrevem a funcionalidade dos requisitos do software, e traduzem naturalmente o que a maioria dos praticantes considera como o objetivo de uma análise orientada a objetos. É importante notar, contudo, que embora os casos de uso sejam mais comumente associados ao desenvolvimento orientado a objetos, a abordagem OO não é pré-requisito para o desenvolvimento dos casos de uso.

Uma das maneiras mais populares de documentar os requisitos de software cai facilmente dentro do desenvolvimento de casos de uso. Trata-se da IEEE Software Requeriments Specification (Especificação de requisitos de Software (SRS)): IEEE 130, largamente usada pelo governo e pela indústria dos EUA. Cada especificação na SRS deveria ser ligada ao elemento do projeto que a referencia, ao segmento de código que a implementa, e aos casos de teste que o verificam, utilizando uma matriz de rastreabilidade de requisitos. [17]

A importância de usar a especificação de requisitos de Software (SRS) é que se descreve o quê o software irá fazer, sem referência ao como as funções serão implementadas. Com a SRS, os casos de uso podem ser claramente documentados e os casos de teste podem ser mapeados para especificar os requisitos.

Um grupo de cenários ou casos de uso são desenvolvidos, para ajudar a identificar os objetos e descrever o comportamento do sistema. Uma vez que o Caso de Uso é concluído, é possível utilizar o Caso de Uso para obter a interação entre os objetos, as mudanças de estado contidas dentro do Caso de Uso, e os requisitos funcionais ou de dados contidos no Caso de Uso.

Exemplos de casos de uso incluem:

- Funcionalidades requeridas para depositar fundos em uma conta bancária.
- Verificar que o limite do cartão de crédito não foi excedido para um pedido de compra.
- Processar uma transação de um cartão de crédito para uma venda.
- Determinar qual rota de uma linha aérea entre duas cidades tomará o menor tempo.

Uma vez que os requisitos do sistema conhecidos e associados aos casos de uso sejam documentados, o resultado é essencialmente uma planta baixa do que o software deve fazer. É importante, neste ponto, quantificar o tamanho dos requisitos da sua planta baixa, usando unidades objetivas de medição de software, tais como Pontos de Função. Muitos modelos de estimativas de projetos, tais como o produto SLIM da QSM de Larry Putnam, o produto PQMPlus da Union Pacific Technologies, o KnowledgePLAN da Artemis, o COCOMOII de Barry Boehm e o Estimate Professional do Software Productivity Center, irão aceitar o ponto de função como uma unidade padrão de medida de tamanho de um projeto. Como todas estes vendedores de soluções e outros especialistas da indústria irão atestar, mensurar o tamanho dos requisitos do usuário é tão crítico para a gerência e o controle do projeto, quanto identificar os requisitos.

Um Caso de Uso é um diálogo entre um usuário e o sistema computacional, com o objetivo de satisfazer uma necessidade específica do usuário. Um Caso de Uso coloca o foco do esforço de desenvolvimento de software naquilo que o usuário precisa que seja feito para que seus objetivos sejam alcançados.

Casos de uso descrevem como os usuários interagem com um sistema computacional.

Normalmente uma tela precisa ser dissecada em transações (entradas externas, consultas externas e saídas externas). Isto é, qualquer tela específica pode conter uma consulta externa (apresentação de informações), saídas externas (dados derivados) e entradas externas (atualizações de arquivos). Os casos de uso fornecem as partes já dissecadas (passos). Um passo pode ser uma transação, um elemento de dado ou

nenhum dos dois. Cada passo precisa ser analisado para determinar se é uma transação ou elemento de dado.

É importante rever o vocabulário utilizado, procurando palavras tais como salvar, atualizar, juntar, calcular e assim por diante.

8.1. Pontos de Função para Mensurar o Tamanho dos Casos de Uso:

Os Pontos de Função mensuram o tamanho da "planta baixa" do software ou o que o software deve fazer, e são independentes de como o software será desenvolvido. Como tal, os Pontos de Função são diretamente compatíveis com as abordagens de SRS e de casos de uso descritos anteriormente.

Uma das maiores barreiras para a contagem de Pontos de Função - a ausência de requisitos documentados do usuário - é resolvida com os casos de uso. Os casos de uso apresentam os requisitos lógicos dos usuários num formato completo e de fácil entendimento, que faz a contagem de Pontos de Função ser absorvida sem problemas.

Usando casos de uso, em conjunto com o modelo lógico de DFD ou modelos de objetos, tem-se uma rota direta para a contagem de Pontos de Função. Os modelos de objetos (ou DFD, se o modelo de objetos não estiver disponível) fornecem um mapeamento dos objetos e classes independentes ou dependentes. Esses objetos (ou grupos de objetos dependentes) mantidos através de serviços padronizados (resumidos por um ou mais casos de uso) são tipicamente contados como Arquivos Lógicos Internos (ALI) em Pontos de Função.

Objetos e classes que são apenas referenciados e que não são mantidos através de funções padronizadas do software (como descrito nos casos de uso) tipicamente transformam-se em Arquivos de Interface Externa (AIE), na metodologia de Pontos de Função. Os casos de uso fornecem, por si só, os processos e funcionalidades necessárias à contagem dos demais componentes da Análise de Pontos de Função: Entradas Externas (EE), Saídas Externas (SE) e Consultas Externas (CE).

Embora um Caso de Uso de forma geral corresponda a uma função lógica em Pontos de Função, isto não é regra geral. Numa situação ideal, deveria haver uma relação um para um entre os casos de uso e as funcionalidades dos Pontos de Função, mas, na prática, um único Caso de Uso pode descrever mais do que uma funcionalidade lógica do usuário.

As pessoas que utilizam Pontos de Função encontrarão maior facilidade para contar Pontos de Função a partir da documentação de casos de uso, diferentemente do caso das misturas de formas de documentação que costumam ser encontrados.

O tamanho do projeto de software em Pontos de Função quantifica o tamanho dos requisitos do usuário e pode ser usado em conjunto com as variáveis físicas e tecnológicas, dentro de um modelo de estimativas, para estimar o esforço, duração, defeitos e outros aspectos do projeto de software.

8.2. Riscos ao Contar Pontos de Função a Partir de Casos de Uso:

Os riscos ao contar pontos de função a partir de casos de uso são normalmente os mesmos existentes na contagem de especificação de sistemas de informação. Isto é, está intimamente relacionada com a qualidade do levantamento dos requisitos:

- Uma transação necessária não foi identificada em um Caso de Uso;
- Uma transação foi erradamente identificada em um Caso de Uso;
- O número de atributos não está claramente definido em um Caso de Uso;
- O número de atributos não pode ser associado a uma entidade.

9. Utilização de Pontos de Função para Estimar Casos de Teste:

Muitas tentativas têm sido feitas para estabelecer uma relação entre pontos de função e o esforço associado ao desenvolvimento de software. Grande parte da dificuldade no estabelecimento desta relação é devida ao fato de ser examinado apenas o relacionamento entre os pontos de função e o ciclo de vida inteiro do software.

A associação entre pontos de função e caso de testes deve passar inicialmente por uma padronização da atividade de testes realizada pela empresa. Não é possível a aplicação de índices de produtividade em atividades que não estejam padronizadas.

9.1. Planejamento, Especificação e Execução dos Testes:

Testes não devem executados aleatoriamente e sim, de uma forma sistemática seguindo-se um plano previamente elaborado, que contenha todas as atividades necessárias à realização. É impossível a aplicação de índices de produtividade em procedimentos não padronizados. [16]

A implantação de um sistema de boa qualidade, dentro de um prazo específico, pode ser seriamente prejudicada caso uma etapa extremamente importante, não seja assim considerada. Nesta etapa são realizados os testes do sistema, onde é realmente verificado se o sistema faz o que se espera que faça de uma forma correta e eficiente.

Os métodos mais comuns de teste realizados são:

- Black-box test: Tem como objetivo testar o sistema sob o ponto de vista de suas funcionalidade. Isto é, se todas as funções necessárias para atender ao usuário foram implementadas;
- White-box test: Também conhecido como teste estrutural. Tem como objetivo validar os dados derivados das funções do sistema;
- Path test: Este teste visa validar todos os caminhos de lógica existentes em cada uma das funções componentes do sistema. Isto é, se todas as regras de negócio foram implementadas corretamente;

- Regression test: Tem como objetivo verificar se as correções solicitadas foram executadas;
- Installation test: Este teste é executado quando da instalação do sistema e tem como objetivo verificar se o sistema ao ser implantado não apresenta problemas quando em funcionamento no ambiente de produção.

9.1.1. Plano de Testes:

No planejamento das atividades de testes são considerados os seguintes tópicos:

- Tipos de testes necessários: unitário (descobrir e corrigir erros), de integração (interface entre módulos), de sistema (volume, stress / performance, interface com o usuário, segurança, interface com outros sistemas) e de aceitação do usuário (atende aos requerimentos, características funcionais);
- Escopo dos testes, o quanto do sistema estará sendo coberto pelos testes. Considere o impacto ou a existência de interface com outros aplicativos / sistemas e os riscos envolvidos. Considere também, as características do ambiente de produção (por exemplo, necessidade de transmissão);
- Avaliar as necessidades de recursos de hardware / software / humanos, armazenamento, configuração do processador e rede de comunicação;
- Estimar esforços e custos para execução dos testes unitários e do sistema;
- Divisão do sistema em módulos;
- Grau de complexidade das rotinas a serem testadas;
- Preparação do ambiente de testes;
- Momento e responsáveis pelo planejamento, execução dos testes e verificação dos resultados;
- Envolvimento de outros setores;
- Forma de geração da massa de dados;
- Existência de documentação de apoio, como por exemplo, casos de testes anteriores;

- Critérios para avaliação dos resultados, ou seja, o que deverá ser atendido para que se considere um teste satisfatório;
- Documentação a ser produzida.

O resultado desta fase deve ser um plano completo com todas atividades incluindo estratégia, custos, prazos e responsabilidades.

9.1.2. Teste Unitário:

Nesta fase o foco é o correto funcionamento de cada uma das aplicações que compõem o sistema de uma forma unitária, não sendo importante o relacionamento entre funções.

As atividades previstas para esta fase são:

- Identificar os programas críticos para testes, baseado em critério como volume, performance e importância;
- Tomar conhecimento, aplicar e difundir entre os componentes da equipe, os padrões técnicos de testes;
- Integrar o responsável pela preparação e execução dos testes com os projetistas do sistema, de forma que se tenha uma boa visão conceitual do sistema;
- Elaborar rotinas especiais para construção da massas dados para teste unitários e do sistema. Podendo ser através da extração de dados do ambiente de produção para o ambiente de desenvolvimento e/ou criando a massa de dados com situações específicas para cada caso de teste;
- Elaborar rotinas especiais de recuperação / restart / commit / rollback para os programas que atualizarem a base de dados, com o intuito de garantir a integridade lógica da Base;
- Definir procedimentos e elaborar rotinas que possibilitem salvar a imagem inicial da base de dados afim de tornar viável a repetição dos testes;
- Elaborar Test Case para cada função componente do sistema, de modo que todas as regras de negócio possam ser validadas.

- Test Case pode ser descrito como sendo um conjunto de passos, que ao ser executado, retornará um resultado prognosticável. O Test Case é desenhado para verificar a funcionalidade de uma função de negócio para um cenário específico. Ele deve descrever passo a passo como a função deve ser testada, contendo inclusive: identificação do Test Case e da função a ser testada, data de criação e de execução, autor, executor, resultado esperado e obtido, input necessário, condição de restrição, fatores de complicação, condições a serem testadas, método de validação e sequência de execução;
- Envolver DBA e AD para que auxiliem a construção de acessos otimizados ao Banco de Dados. É de fundamental importância a análise dos acessos ao banco, como por exemplo, avaliação de necessidade de criação / alteração / exclusão de índices, alteração / união / divisão de tabelas ou até mesmo, ajustes necessários no ambiente do banco;
- Testar exaustivamente cada função, usando a massa de dados construída e os "Test Case" criado;
- Registrar no "Test Case" todos os fatos ocorridos nos testes, independente de terem sido bem sucedidos ou não;
- Avaliar os resultados (esperado x realizado);
- Solicitar a revisão do resultado dos testes pelo Grupo de Qualidade.

A execução dos testes unitários é um procedimento cíclico (teste, correção, teste, correção, teste...). Um bom planejamento, eficiência e dedicação dos responsáveis pelas correções e ajustes necessários e a utilização correta de "Test Case", serão fatores críticos de sucesso para se obter ao final desta fase um produto 100% testado e corrigido.

Ao final dos testes unitários deve-se fazer uma reunião formal para aprovação dos resultados.

9.1.3. Teste do Sistema:

Esta etapa visa testar as funções do sistema de forma integrada, simulando o processamento total das rotinas, módulos e o sistema como um todo. Tem como objetivo assegurar a qualidade, a funcionalidade e desempenho da aplicação.

As atividades previstas para esta etapa são:

- Avaliar a possibilidade da participação de pessoas independentes (fora da equipe do projeto) para a realização dos testes;
- Avaliar e validar o plano de testes e o resultado dos testes unitários. Caso sejam necessários ajustes e alterações, executá-las e validá-las com o Gerente do Projeto;
- Elaborar Test Script de modo a grupar Test Case, para cada uma das sequências operacionais que o sistema será operacionalmente utilizado;
- Test Script descreve como um conjunto de Test Case deve ser ordenado, de modo a representar o encadeamento operacional de cada uma das sequências lógicas de negócios. Ele deve conter inclusive: identificação do Test Script, responsável pela criação, execução e validação, data de criação e de execução, fluxo operacional a ser testado, objetivo, informações sobre a operação, identificação dos "Test Case" envolvidos, pré-requisitos, dados de teste, resultado esperado e obtido, forma de validação e documentação em anexo como por exemplo, relatórios emitidos;
- Selecionar e preparar previamente os dados de modo que todas as situações possíveis, possam ser testadas;
- Verificar se todos os arquivos / tabelas a serem utilizados já foram criados, carregados e estão disponíveis. Ao utilizar arquivos / tabelas de outros sistemas, verificar se o conteúdo atende aos testes;
- Solicitar a área de DBA a criação de acessos conforme os perfis desejadas;
- Procurar testar o sistema em módulos e exaustivamente utilizando-se dos formulários de "Test Case" e "Test Script" elaborados;
- Documentar todas as ocorrências nos formulários de "Test Case" e "Test Script", reportando a quem de direito.
- Verificar a aderência dos resultados;

- Obter o parecer do Grupo de Revisão do Projeto / Grupo de Qualidade sobre os testes de programas críticos;

9.1.4. Teste Integrado e Teste de Aceitação:

Esta fase tem como objetivos desenvolver as atividades relacionadas com o teste integrado e teste de aceitação do sistema. Corresponde ao nível físico de concepção para o nível físico de operação. A execução desta fase deve suceder a aprovação formal dos testes unitários e de sistema. Isto é, o teste integrado e de aceitação do sistema só poderão ser realizados com as funções 100 % implementadas e testadas.

As atividades previstas para esta etapa são:

- Elaborar o plano para teste integrado e teste de aceitação. Os formulários de Test Case e Test Script utilizados, são também perfeitamente válidos para esta etapa;
- Caso seja viável, estes testes poderão ser realizados no próprio local do usuário de modo a verificar adicionalmente, se o sistema opera naquele ambiente;
- Avaliar as necessidades de recursos de hardware / software / humanos, armazenamento, configuração do processador e rede de comunicação;
- Estimar esforços e custos para execução dos testes;
- Definir os critérios de aceitação do usuário, do ciclo de testes, descrever os dados para testes, teste de auditoria, roteiros de testes e infra-estrutura necessária;
- Elaborar a massa de dados completa, que seja capaz de cobrir todos os casos que compõem os requisitos de negócio do usuário. Esta massa de dados já deve ter sido elaborada com o intuito de atender às necessidades dos testes anteriores;
- Elaborar o cronograma inicial contemplando todas as atividades necessárias;
- Executar os mesmos tipos de testes realizados durante o teste do sistema. A diferença entre estes dois testes é o objetivo, pois no teste do sistema busca-se descobrir defeitos enquanto que no teste de aceitação, busca-se demonstrar também que os critérios de aceitação foram atendidos;

- O roteiro deve também ser executado levando em consideração o volume, desempenho, controle & segurança de acesso, recuperação / contingência;;
- Aprovar os resultados dos testes de aceitação em reunião formal;

9.1.5. Comprometimento dos Envolvidos:

Na execução da fase de testes o comprometimento de todos os envolvidos (usuário, gerência, grupo de revisão de qualidade, desenvolvedor, etc.), é fator crítico de sucesso para a obtenção de um produto de qualidade dentro do prazo e custo estabelecido.

Alguns pontos extras podem ser considerados quando do planejamento, especificação e execução dos testes:

- É extremamente inconveniente relatar o mesmo erro várias vezes e verificar que a correção não está sendo realizada conforme solicitada ou que não foi testada corretamente antes de ser disponibilizada. O implementador tem obrigação de testar exaustivamente uma correção solicitada antes de entregá-la ao responsável pelos testes. Isto influencia diretamente no bem estar da equipe do projeto e nos índices de produtividades obtidos.
- Estudos e registros históricos relatam que considerando-se as produtividades realizadas em um projeto de sistema em termos de número de pontos de função por unidade de tempo, os percentuais de tempo gasto na fases de um projeto, podem ser em média assim distribuídos[16]:
    - 2% Elaboração Business Case
    - 21% Definição dos Requerimentos de Negócios
    - 52% Implementação Física das Funções
    - 20% Testes
    - 5% Implantação
- Observa-se no acima exposto, que o tempo de teste em relação ao tempo total do projeto, corresponde em média a 20 %. Aplicando-se este percentual ao custo total de desenvolvimento de um projeto, desconsiderando-se o custo de correção dos

problemas encontrados, nota-se que esta parcela equivale a uma quantia razoável de dinheiro. Quando se incorpora os custos relativos à correção dos erros encontrados, a parcela relativa a fase de testes como um todo, seria somente menor que a de implementação física das funções. Em alguns casos, podemos ter algo em torno de 1/3 do custo de desenvolvimento do sistema [16].

- Documente em ata e formulários específicos todos os passos e fatos ocorridos durante os testes. O formalismo adotado não deve ser considerado como um processo burocrático. Existem vários benefícios obtidos diretamente com a formalização, como por exemplo: faz com que as pessoas prestem mais atenção no que está sendo realizado, possibilita rastreabilidade das condições de erro, permite definir indicadores de erros como número de erros por ponto de função, alem de servir como histórico para outros projetos.
- Por último, repetindo uma frase já dita várias vezes neste documento, "não é possível a aplicação de índices de produtividade em atividades que não estejam padronizadas".

9.2. Casos de Teste:

A relação entre pontos de função e o número de casos de teste é muito forte. Capers Jones estima que o número de casos de teste é aproximadamente igual ao número de Pontos de Função elevado a 1,2 ($PF^{1,2}$). Isto é, os casos de teste crescem a uma taxa mais rápida do que os pontos de função. Este é um fato intuitivo, porque conforme um aplicativo cresce, os interrelacionamentos nele contidos tornam-se mais complexos.

Por exemplo, se um aplicativo em desenvolvimento tiver 1.000 pontos de função, deverão existir aproximadamente 4.000 casos de teste. Obviamente, a equipe de desenvolvimento deveria começar a criar os casos de teste tão rapidamente quanto possível. Se a equipe esperar até que a codificação tenha terminado, provavelmente serão criados muito menos do que 4.000 casos de teste.

Há um grande benefício em estimar o número de casos de teste. Entender o número de casos de teste leva à análise lógica de comparar os casos reais com os esperados. Se os casos reais forem em quantidade inferior à esperada, a cobertura de testes será inadequada. A cobertura de testes é uma indicação direta dos defeitos potenciais e custos futuros com manutenção. Quão maior for a distância entre a quantidade esperada e a real de casos de teste, maior o potencial de defeitos não detetados durante a respectiva fase de testes.

O único caminho para elevar a qualidade do software em produção é se e somente se todo o software migrado para a produção for de qualidade superior ao software atualmente em produção. É importante entender tanto os defeitos potenciais do software de produção, quanto os defeitos potenciais do novo software. Pode ser impossível estimar o número de defeitos embutidos no software atualmente em produção, mas é possível estimar o potencial de defeitos para cada nova versão do software. A diferença nas quantidades real e estimada dos casos de teste é um bom indicador do potencial de defeitos.

Taxas de produtividade inconsistentes entre diferentes projetos podem ser uma indicação de que não está sendo seguido um processo padrão. A produtividade é definida como a razão entradas / saídas. No caso de software, a produtividade é definida como a quantidade de esforço requerida para entregar um certo conjunto de funcionalidades (medidas em pontos de função).

10. Mapeamento de Casos de Teste em Casos de Uso e Pontos de Função:

Se todos os requisitos do sistema forem mapeados em casos de teste, e todos os casos de teste forem executados, seria razoável supor que o sistema alcançaria os resultados desejados. Embora este objetivo possa ser alcançado através do uso de modelos de Orientação a Objetos, a conversão dos modelos para uma linguagem simples pode reduzir a ambigüidade e ajudar a definir claramente os requisitos do sistema. Os casos de uso fornecem tal grupo de documentos, escritos em uma linguagem simples.

Os casos de teste podem ser mapeados para componentes da Análise de Pontos de Função e, através do uso de modelos de estimativas, uma estimativa do tempo para os testes pode ser derivada. Algumas empresas utilizam como medida padrão considerar o percentual da funcionalidade que foi testado (em PF), dividido pelo tamanho total do projeto.

Sempre que é introduzida alguma alteração de escopo no projeto, ou que são identificados novos requisitos (previamente classificados como *Desconhecidos*, *Imprevisíveis*), novas documentações de SRS e casos de uso devem ser elaboradas, para facilitar a medição das alterações em Pontos de Função. Inserindo o tamanho revisto do projeto em Pontos de Função dentro do modelo de estimativas, o impacto do escopo da alteração em termos de recursos, planejamento e orçamento poderá ser propriamente estimado. Se as alterações de escopo forem subseqüentemente aceitas no projeto, os casos de uso, novos ou revistos, deveriam ser escritos, os detalhes dos PF verificados, e casos de teste desenvolvidos para abordarem as funcionalidades novas ou revistas. A importância dos casos de teste para as funcionalidades novas ou revistas é tão crítica para a qualidade do produto desenvolvido quanto os casos de teste originais.

Gerenciar as alterações de software desta forma é similar à maneira como uma construção de um prédio é gerenciada -- solicitações de alteração são dimensionadas em termos do seu impacto no projeto. As modificações de recursos e custos são

estimadas e então uma decisão é tomada, sobre como e de que forma fazer. O sistema de desenvolvimento de projetos resultará numa maior qualidade de software quando as alterações introduzidas durante o projeto forem planejadas, estimadas e controladas, e não mais chegarem com a ordem "façam".

Através de casos de uso, especificações de requisitos padronizadas, casos de teste e Pontos de Função, os projetos de software poderão maximizar o número de requisitos "conhecidos" nas etapas iniciais do ciclo de desenvolvimento.

A utilização de casos de uso requer que a equipe de projeto aumente o tempo gasto nos estágios iniciais do desenvolvimento do software, planejando e garantindo que os requisitos estejam completos, bem definidos e documentados. A contagem de Pontos de Função freqüentemente mostrará ausência de clareza nos casos de uso, podendo ajudar a eliminar a ambigüidade onde a mesma se tornar aparente, porque a metodologia de Pontos de Função vincula os repositórios de dados (objetos, entidades) com a sua manipulação e uso dentro das funções do software. Os casos de teste baseados em casos de uso garantem que os testes sejam realizados para cada caso de teste do usuário e, quando os casos de uso forem desenvolvidos baseados em SRS, os mesmos irão garantir que todos os casos de uso apontarão para os requisito originais do usuário.

Através da documentação das alterações à medida que elas ocorrerem através de SRS, casos de uso, casos de teste e, ainda, com a atualização da contagem do projeto em Pontos de Função, a documentação e o controle do desenvolvimento do projeto tornar-se-ão mais fáceis. O impacto das alterações introduzidas posteriormente no ciclo de vida do desenvolvimento pode ser controlado antes que ciclos de re-trabalho sejam alimentados e o que projeto derive para fora do controle.

11. Impacto das Novas Tecnologias:

Embora a função primária de todos os aplicativos de software seja o alcance dos objetivos negociais, tais aplicativos existem sob todas as formas e tamanhos. As definições do IFPUG necessitam de modificações, para que possam ser utilizadas no mundo moderno de hoje. As Diretrizes Para Contagem do IFPUG (IFPUG Counting Guidelines) não sofreram modificação por quase oito anos. Nesse período, o mundo da tecnologia deu um salto à frente, enquanto as Regras Para Contagem de Pontos de Função permaneceram estáticas. É importante ser capaz de adaptar as Regras de Contagem do IFPUG (mantendo a conformidade com os padrões do passado) ao mundo de hoje, com seu ritmo rápido e sempre passando por mudanças [18].

11.1. Definições Melhoradas Para Pontos de Função:

É difícil aplicar as definições do IFPUG às novas tecnologias. Essas definições tornam-se muito mais claras a partir de algumas simples e poucas alterações. As definições seguintes não devem afetar a contagem de pontos de função, porém reduzem a curva de aprendizagem na aplicação de pontos de função a tecnologias novas e emergentes:

11.1.1. Entrada Externa (EE):

- **Definição Melhorada:** é um processo elementar no qual dados atravessam a fronteira de fora para dentro. Tais dados podem vir de uma tela de entrada de dados, por via eletrônica ou através de um outro aplicativo. Os dados podem ser informações de controle ou informações do negócio. No caso dos dados serem informações do negócio, serão utilizados para manter um ou mais arquivos lógicos internos. Se os dados forem informações de controle, não será necessário que atualizem um arquivo lógico interno.
- **Definição do IFPUG 4.0**: processam dados ou informações de controle procedentes de fora da fronteira do aplicativo. A própria entrada externa é um processo elementar. Os dados processados mantêm um ou mais arquivo lógico

interno (ALI). As informações de controle processadas podem ou não manter um arquivo lógico interno (ALI).

A definição do IFPUG diz "de fora da fronteira do aplicativo". Como a mesma definição afirma que "os dados processados mantém um ou mais ALI", fica claro que a informação vem de fora para dentro da fronteira. Isto é importante por uma série de razões. Fica claro, a partir da definição melhorada, que valores calculados armazenados são elementos de dados para a entrada externa, mas valores calculados não armazenados não são elementos de dados para a entrada externa. Isto é verdade porque o valor calculado que não é armazenado não atravessou a fronteira (de fora para dentro) e não mantém um ALI.

Em um ambiente GUI ou OO é comum que a informação se mova de uma janela para a próxima. O movimento de dados em si não é considerado uma entrada externa, pois não atravessou a fronteira da aplicação (de fora para dentro) e não mantém um ALI.

11.1.2. Saídas Externas (SE):

- **Definição Melhorada:** um processo elementar no qual dados derivados passam através da fronteira, de dentro para fora. Os dados criam relatórios ou arquivos de saída, que são enviados a outros aplicativos. Esses relatórios e arquivos são criados a partir de um ou mais arquivos lógicos internos e/ou arquivos de interface externa.

  Dados derivados são dados cujo processamento vai além da recuperação e edição direta de informações de arquivos lógicos internos ou arquivos de interface externa. São o resultado de algoritmos e/ou cálculos. Dados derivados ocorrem quando um ou mais elementos são combinados com uma fórmula, de modo a gerar ou derivar um ou mais elementos de dados adicionais. Um algoritmo é definido como um procedimento mecânico para executar um dado cálculo ou resolver um problema utilizando uma série de passos.

- **Definição IFPUG 4.0:** Uma Saída Externa (SE) é um processo elementar que gera dados ou informações de controle, enviados para fora da fronteira do aplicativo.

O manual IFPUG 4.0 não fornece uma definição para dados derivados, nem diz onde a informação estava antes de ser enviada para fora da fronteira do aplicativo. Se a informação é enviada para fora da fronteira, é seguro dizer que ela estava dentro da fronteira. Como a informação estava dentro da fronteira, ela deve estar contida em um arquivo lógico interno (ALI) ou arquivo de interface externa (AIE).

11.1.3. Consultas Externas (CE):

- **Definição Melhorada:** é um processo elementar com componentes de entrada e saída, que resulta na recuperação de dados de um ou mais arquivos lógicos internos e/ou arquivos de interface externa. A informação recuperada é enviada para fora da fronteira do aplicativo. O processo de entrada não atualiza nenhum Arquivo Lógico Interno e o lado de saída não contém dados derivados.
- **Definição IFPUG 4.0:** é um processo elementar constituído por uma combinação entrada-saída que resulta na recuperação de dados. O lado de saída não contém dados derivados. Nenhum arquivo lógico interno é mantido no processamento

A Definição do IFPUG não é clara por uma série de razões. A definição do IFPUG diz "resulta na recuperação de dados". Se é este o caso, os dados devem ser recuperados de algum lugar dentro da fronteira do aplicativo. O único lugar onde os dados podem residir é um ALI ou AIE. Neste caso, a definição deveria dizer explicitamente "de um ALI ou AIE".

A Definição IFPUG não declara explicitamente que a informação deve ser enviada para fora da fronteira da aplicação (como no caso de uma SE). Isto é importante para OO, porque os objetos se comunicam uns com os outros. Apenas quando um objeto envia alguma coisa para fora da fronteira é que ele pode ser considerado uma consulta externa.

Adicionalmente, o manual IFPUG não faz distinção clara entre uma SE e uma CE. É comum nos ambientes OO e GUI que uma SE tenha um lado de entrada. O

único fator que permite distinguir uma da outra é que uma CE não pode ter dados derivados. Se este é o caso, uma SE obrigatoriamente possui dados derivados (senão vai ser uma CE).

11.1.4. Arquivo Lógico Interno (ALI)

- **Definição Melhorada:** um grupo lógico de dados relacionados, identificável pelo usuário, que reside inteiramente dentro da fronteira do aplicativo e é mantido através de Entradas Externas.
- **Definição IFPUG:** é um grupo lógico de dados relacionados, identificável pelo usuário, ou informações de controle mantidas dentro da fronteira do aplicativo.

A diferença primária está nas últimas palavras da definição. A diferença é a expansão do que se quer dizer com "mantidas". A única coisa que mantém um arquivo lógico interno é uma Entrada Externa.

11.1.5. Arquivo de Interface Externa (AIE)

- **Definição Melhorada:** um grupo lógico de dados relacionados, identificável pelo usuário, que é utilizado apenas para referência. Os dados residem inteiramente fora do aplicativo e são mantidos por um outro aplicativo. O Arquivo de Interface Externa é um Arquivo Lógico Interno para outro aplicativo.
- **Definição IFPUG:** é um grupo de dados relacionados, identificável pelo usuário, ou informações de controle, referenciados pelo aplicativo, porém mantidos dentro da fronteira de um outro aplicativo. Isto significa que um AIE contado para um aplicativo deve ser um ALI para outro aplicativo.

Não tem diferença significativa nas definições.

11.2. Entendendo Aplicações Internet/Intranet:

A grande maioria dos sites da web contém nenhum ou muito poucos pontos de função. A maioria dos sites nada mais são do que folhetos online. Isto é, nenhum

arquivo é mantido e o conteúdo das páginas não é o resultado da leitura de um arquivo.

Por outro lado, há aplicações Internet/Intranet que oferecem funcionalidade ligada ao negócio e precisam ser consideradas. Cada vez mais organizações estão utilizando aplicações Internet para implementar funcionalidade essencial do negócio. A utilização de pontos de função com este tipo de aplicativo apresenta alguns problemas específicos. O primeiro deles é a fronteira. O segundo é a identificação das transações, dos arquivos lógicos internos e/ou arquivos de interface externa.

A fronteira para um aplicativo Internet é definida de maneira semelhante à utilizada para aplicativos tradicionais. No caso de aplicativos tradicionais, a fronteira não é desenhada simplesmente ao redor da interface com o usuário ou de um grupo de telas, mas do aplicativo inteiro. Com freqüência os aplicativos Internet são apenas extensões de aplicativos existentes. Há uma tendência a se "criar" um aplicativo para a extensão Internet, mas esta abordagem não é correta.

Transações (entradas externas, saídas externas, consultas externas) seguem as mesmas regras básicas utilizadas em aplicativos convencionais.

Se uma entrada externa não for uma entrada de controle, então deverá atualizar um arquivo lógico interno. Um exemplo de uma entrada externa seria algo como "submeter formulários". Os formulários submetidos podem ser um texto html atualizado com informações. Por exemplo, eu tive uma série de pesquisas em meu site na web. A cada resposta, a pesquisa survey.htm era atualizada. Este é um exemplo de uma entrada externa. As informações atravessaram a fronteira de fora para dentro e um arquivo lógico interno foi mantido.

Alguns aplicativos permitem a atualização online de arquivos temporários. Tais arquivos temporários são utilizados para atualizar arquivos permanentes. Embora a entrada externa esteja atualizando um arquivo temporário, este arquivo temporário é logicamente o mesmo que o arquivo permanente. Isto significa que o arquivo temporário é uma imagem espelho ou subconjunto do arquivo permanente.

Da mesma forma que uma consulta externa, uma saída externa pode ter e com freqüência tem um lado de entrada. As informações são lidas de um arquivo lógico interno ou de um arquivo de interface externa.

Da mesma forma que em uma saída externa, as informações em uma consulta externa precisam ser lidas de arquivos lógicos internos ou arquivos de interface externa.

Sites da web desenvolvidos com a utilização de FrontPage e outras ferramentas de html podem ou não conter funcionalidade. A chave é entender onde as informações residem e como são processadas. A grande maioria dos sites web nada mais são do que apenas menus e texto.

11.3. Aplicativos GUI:

Da mesma forma que os aplicativos tradicionais, os aplicativos GUI apresentam as informações de duas maneiras, como Consultas Externas ou como Saídas Externas. Uma consulta externa é uma combinação entrada-saída que resulta na recuperação de informações de um ou mais arquivos. O lado de saída não contém dados derivados e nenhuma conclusão é tirada. Por exemplo, um usuário pode querer visualizar o último pedido de um cliente. O nome do cliente seria introduzido como o critério de pesquisa, sendo então mostrado o último pedido. Neste caso, o lado de saída não contém dados derivados e nenhuma conclusão é tirada.

Por outro lado, uma Saída Externa contém dados derivados. As informações são processadas com a utilização de algoritmos e algum tipo de conclusão é apresentado. Um erro comum é contar toda a parte online como CE e toda a parte enviada ao papel como SE. Tanto CE quanto SE podem ser mostrados online ou em hard copy.

Utilizando a definição estrita de elemento de dado constante do Manual de Práticas de Contagem do IFPUG, teremos “Um Elemento de Dado é um campo não recursivo, identificável pelo usuário”. Infelizmente esta definição não provê suficiente orientação na contagem de aplicativos GUI. Na verdade, o Manual de Práticas de

Contagem do IFPUG não oferece detalhes a respeito de Botões de Rádio, Caixas de Verificação, Caixas de Listagem, Caixas de Combinação e assim por diante. Apesar disso, um elemento de dado é uma informação que é armazenada em um Arquivo Lógico Interno, ou utilizada para chamar uma transação.

Saídas externas podem transmitir informações textuais, gráficas ou eletrônicas. Avalia-se uma saída externa combinando-se o número de tipos de arquivos referenciados e o número de tipos de elementos de dados únicos (não recursivos).

Elementos gráficos são contados da mesma maneira que as SE textuais, isto é, o elemento gráfico é avaliado com base no número de TED (Tipos de Elementos de Dados) e no número de RTA (Referências a Tipos de Arquivos). Na verdade, informações recursivas são facilmente identificáveis em um elemento gráfico, sendo às vezes mais difíceis de visualizar em um relatório textual.

Consultas Externas são muito comuns em aplicativos GUI. Conforme explicado anteriormente, CE não contém dados derivados ou calculados. Uma caixa de listagem dinâmica (drop down box) é um exemplo de Consulta Externa. É simplesmente uma leitura de um arquivo, como no caso de uma caixa de listagem oferecendo uma lista de nomes de países. A caixa de listagem seria contada como uma CE, caso os nomes dos países estivessem contidos em um arquivo lógico interno, ou em um arquivo de interface externa.

Uma consulta serial é uma consulta seguida por outra. Por exemplo, o usuário pode escolher um nome de país de uma caixa de listagem (primeira CE), o qual pode ser usado como entrada para uma segunda CE, que traz os detalhes do país selecionado.

Em um menu dinâmico, como, por exemplo, o do Word, são mostrados os últimos arquivos que foram abertos. Pode-se facilmente concluir que esta informação está sendo lida de algum tipo de arquivo interno. Por essa razão, a informação é dinâmica. O menu seria contado como uma consulta externa.

Embora o Manual do IFPUG diga explicitamente que menus não são contados, neste caso fica claro que o menu é dinâmico e se modifica.

A distinção real é se o menu é dinâmico ou estático, isto é, se o conteúdo da tela ou relatório é dinâmico (lido de algum arquivo), ou estático.

Uma Entrada Externa é o método através do qual as informações em um ALI são mantidas (incluídas, alteradas ou excluídas). Os aplicativos GUI costumam ter Entradas Externas precedidas por consultas seriais. Um usuário pode selecionar um nome de cliente de um listbox (1ª CE), sendo o nome utilizado como entrada para a 2ª CE, a qual retorna endereço, telefone, CEP e outras informações. A partir deste ponto, o usuário pode incluir, alterar ou excluir as informações do cliente (3 EE). Neste caso, esta única tela representaria 2 CE e 3 EE.

Entradas de controle alteram o comportamento de um aplicativo ou o conteúdo de um relatório. Na tela de controle "Criar Relatório", o usuário pode selecionar quais relatórios deverão ser produzidos. Esta tela possui diversos tipos de elementos de dados: caixa de verificação, elemento gráfico, dimensões, elementos, sub-itens e teclas de ação.

Nota-se que o usuário pode escolher cada relatório individualmente. De fato, cada relatório é um objeto. O relatório gerado é uma combinação de diversos relatórios (ou objetos). Cada objeto tem vários atributos.

Considera-se um aplicativo escrito para um único idioma. É provável que os cabeçalhos de relatórios e descrições textuais estejam todos codificados internamente nos programas, isto é, o usuário não consegue alterar dinamicamente os cabeçalhos ou o texto. Agora, considerando-se um aplicativo que tenha sido desenvolvido com a previsão de múltiplos idiomas. Os cabeçalhos dos relatórios e as descrições textuais serão todos lidos de arquivos.

Se as saídas externas estiverem disponíveis em múltiplos idiomas, várias coisas precisam ser consideradas. Primeiro provavelmente deverá haver alguma entrada de controle que permitirá ao usuário selecionar dinamicamente o idioma. Segundo, há uma RTA adicional, que contém o texto no idioma selecionado. Terceiro, o arquivo interno lógico correspondente a este idioma é mantido através de uma entrada externa. Quarto, há mais elementos de dado no relatório. Se uma saída externa

estiver disponível em mais de um idioma, então a mesma não será considerada uma saída externa única, mas será mais complexa (mais TED e mais RTA).

Botões de Rádio são tratados como tipos de elementos de dado. Em um grupo de botões de rádio, o usuário tem a opção de selecionar somente um botão de rádio de cada vez, de modo que um único tipo de elemento de dado é contado para todos os botões de rádio contidos no grupo inteiro. Na criação do relatório "webtrends", os botões de rádio "2 D" e "3 D" representam apenas um único elemento de dado.

Botões de comando podem especificar uma ação de inclusão, alteração, exclusão ou consulta. De acordo com as regras de contagem do IFPUG, cada botão de comando seria contado como um Tipo de Elemento de Dado (TED) correspondente à ação por ele comandada. Por exemplo, um simples aplicativo para acompanhar Distribuidores poderia ter campos para Nome do Distribuidor, Endereço, Cidade, Estado, CEP, Telefone e Fax. Isto representaria sete elementos de dado (7 TED) e o botão de comando de inclusão representaria o oitavo elemento de dado. Resumindo, a entrada externa de inclusão representa uma entrada externa com oito elementos de dado, a entrada externa de alteração representa outra entrada externa com oito elementos de dado (7 campos mais o botão de comando de alteração) e a entrada externa de exclusão representa a última entrada externa, com oito elementos de dado (7 campos mais o botão de comando de exclusão).

A apresentação de uma imagem gráfica é simplesmente mais um elemento de dado. Um aplicativo de estoque pode conter dados a respeito de peças. Pode conter nome da peça, fornecedor, tamanho, peso e incluir uma imagem esquemática da mesma. Esta imagem é tratada como um outro elemento de dado.

Um outro exemplo seria um mapa. O mapa pode ser "quente", isto é, conforme o mouse é movido sobre o mapa, diferentes nomes de cidades são mostrados. Se o usuário clicar sobre um ponto específico do mapa, aparecerão detalhes a respeito da cidade selecionada. Se os detalhes acerca de cada cidade estiverem contidos em um arquivo lógico interno, ou em um arquivo de interface externa, os detalhes poderão ser considerados uma consulta externa.

Muitos aplicativos GUI têm um byte de som anexado. Isto representa um elemento de dado. O número de notas tocadas é simplesmente informação recursiva. Se o tamanho do byte de som aumentar, ainda assim continuará representando um único elemento de dado. Se o Hino Nacional for tocado durante dois ou quatro segundos, ainda assim teremos apenas um elemento de dado. Quanto mais tempo o Hino tocar, mais "informação recursiva" estará presente.

Uma imagem fotográfica é um outro exemplo de elemento de dado. Um aplicativo de recursos humanos pode mostrar o nome do empregado, data de admissão, etc., bem como uma fotografia do empregado. A fotografia será tratada da mesma forma que o nome ou a data de admissão. É uma outra informação a respeito do empregado. A fotografia é armazenada e mantida como qualquer outra informação a seu respeito.

Há três tipos de mensagens geradas em uma aplicação GUI: Mensagens de Erro, Mensagens de Confirmação e Mensagens de Notificação. Uma mensagem de erro e uma mensagem de confirmação indicam que um erro aconteceu, ou que um processo foi ou será completado. Uma mensagem do tipo "Por favor informe o CEP" seria um exemplo de mensagem de erro. Uma mensagem do tipo "Confirma a exclusão do cliente?" é um exemplo de mensagem de confirmação. Nenhum desses tipos de mensagem é tratado como uma Saída Externa única. São tratados como elementos de dado para a transação apropriada.

Por outro lado, uma mensagem de notificação é uma mensagem do negócio. É a base para o processamento e para que uma conclusão seja obtida. Por exemplo, você pode tentar retirar de um caixa automático mais dinheiro do que o disponível em sua conta, recebendo a temida mensagem "Saldo insuficiente para esta transação". Isto é o resultado obtido a partir de informações lidas de um arquivo a respeito de seu saldo atual, com uma conclusão delas decorrente. Uma mensagem de notificação é tratada como uma Saída Externa.

Mensagens de Notificação podem ser resultantes de processamento não visto pelo usuário. Se uma mensagem for criada para envio a um pager em um dado momento, isso funcionará como um alarme, isto é, a hora corrente será comparada

com a hora prevista para a mensagem e, quando ambas forem iguais, a mensagem será enviada. A mensagem de pager tem um único elemento de dado, o texto da mensagem.

11.4. Orientação a Objeto:

Na visão orientada a objetos dos sistemas de software existe uma única entidade, chamada objeto, a qual representa tanto os dados quanto os procedimentos. Os objetos podem ser manipulados da mesma forma que os dados. Contudo, assim como os procedimentos, os objetos também podem descrever manipulações.

Um objeto é uma coleção de dados (atributos e propriedades) e lógica funcional (métodos). Os dados definem o estado do objeto e os métodos, o seu comportamento. Há dois tipos de métodos: primeiro, os métodos de interface, os quais oferecem um meio de comunicação com o objeto; segundo, os métodos internos, que criam comportamentos para o objeto, mas não são acessíveis de fora do mesmo. O método de interface provê uma forma clara e definida para comunicação com o objeto. O método interno precisa ser conhecido apenas pelo projetista do objeto. Os usuários do objeto precisam conhecer APENAS os métodos de interface.

Pontos de Função medem software através da quantificação da funcionalidade entregue ao usuário, com base principalmente no desenho lógico. O termo usuário final ou usuário é freqüentemente utilizado sem que se especifique de quem se trata. No caso aqui considerado, o usuário é um usuário sofisticado, alguém que entende o sistema de um ponto de vista funcional e que provavelmente fornece os requisitos ou efetua o teste de aceitação. Pontos de Função tornam possível a mensuração do tamanho dos sistemas de informações de acordo com o que o usuário vê e com o que interage. Esta definição é semelhante à de método de interface anteriormente apresentada.

Todos os métodos internos existem para dar suporte a métodos de interface. Os métodos internos não são um fim em si, mas mecanismos que apoiam o método de interface em sua tarefa de entregar informações ao usuário.

Um método é simplesmente a ação causada por uma mensagem. Métodos são as coisas que um objeto pode fazer. Métodos operam sobre os dados contidos no objeto, modificando-os ou consultando-os.

Atributos nada mais são do que características de um objeto.

Como os objetos são definidos? Um objeto é definido através de sua classe, a qual determina tudo a seu respeito. Objetos são instâncias de uma classe. Tudo o que você precisa fazer é criar uma subclasse da classe original. Esta nova classe herda todas as mensagens existentes e, dessa forma, herda todo o comportamento da classe original. A classe original é chamada a classe mãe, ou superclasse da nova classe. Um pouco mais de jargão – diz-se que uma subclasse é uma especialização da superclasse e, por outro lado, que a superclasse é uma generalização de suas subclasses. Tudo isto é de fato independente da contagem de pontos de função. O importante é dirigir a atenção para aquilo que cada objeto realmente faz.

Os objetos se comunicam uns com os outros através de troca de mensagens. Uma mensagem contém um nome que identifica seu destino, podendo conter também alguns argumentos ou parâmetros. A mensagem, quando recebida, causa a execução do método apropriado no objeto de destino. Na sua forma mais simples, uma mensagem é o mesmo que uma chamada de função ou procedimento no sentido tradicional, uma vez que o efeito líquido é o mesmo. Muitas vezes o objeto que envia a mensagem é chamado de servidor e o objeto que recebe a mensagem é chamado de cliente. Uma mensagem preparada e enviada ao cliente geralmente vai se manifestar como Saída Externa ou Consulta Externa. Uma mensagem preparada e enviada do cliente ao servidor é geralmente uma Entrada Externa.

Uma mensagem é uma solicitação para que um objeto execute uma seqüência de ações através de um ou mais métodos chamados. Um método é uma unidade de lógica funcional contida em um objeto. De acordo com os padrões do IFPUG, uma transação deve possuir lógica de processamento única e que represente a menor unidade de atividade significativa para o usuário do negócio. Como um método é uma unidade de lógica funcional contida em um objeto, é bastante semelhante à definição de uma transação – de acordo com as Regras de Contagem do IFPUG.

Nas aplicações OO, somente dados persistentes são considerados Arquivos Lógicos Internos e não os dados transientes. Dados transientes são válidos somente dentro de um programa ou transação, sendo perdidos quando o programa ou transação termina. Na programação tradicional, este caso seria semelhante ao de um arquivo temporário. Os dados persistentes são armazenados fora do programa e sobrevivem ao seu término.

Dados persistentes são armazenados em Bancos de Dados Relacionais ou Bancos de Dados OO. Com freqüência, os dados contidos em terceira forma normal representarão um arquivo lógico interno. A terceira forma normal procura remover quaisquer dependências entre os atributos não chaves. Para ser considerado um arquivo lógico interno (ALI), o arquivo (ou tabela) deve ser um agrupamento lógico de dados, mantido ou modificado através de processos elementares (métodos).

12. Gerenciamento de Projetos de eCommerce:

Quais as diferenças entre o gerenciamento de projetos convencionais de TI e um projeto de eCommerce? O que muda? Alguém sabe?

Os projetos de e-Commerce possuem algumas particularidades não comuns nos típicos projeto de sistemas de informações. A flexibilidade, agilidade e rapidez no desenvolvimento são fundamentais para o sucesso do projeto. Porém, essas três características influenciam significantemente o processo padronizado de desenvolvimento de sistemas e por conseqüência direta, dificulta a aplicação de uma métrica para estimativa de tamanho de sistemas. Fato é que:

- Os projetos são mais críticos para o negócio (prazos mais curtos, erros mais caros, mais fácil perder o emprego);
- Os projetos são mais visíveis (o desenvolvimento de sistemas saiu da cozinha e foi para o salão do restaurante, sendo dada maior importância ao conhecimento do negócio em relação à técnica, com nova ênfase no "estilo" empresarial);
- A disponibilidade é muito importante (o objetivo é ter 99,9999% de disponibilidade, os "seis noves");
- Segurança - Disruption Of Service, ou quebra no serviço, algo a ser evitado a todo o custo);
- Simplifique os processos de negócio que impactem o cliente;
- Mantenha uma visão de 360 graus do relacionamento com o cliente;
- Escalabilidade - posso crescer?

Jack Duggal, da Inacom Corporation e da University of Hartford, escreveu em seu livro “e-PM Paradox” (o paradoxo do e-Gerenciamento de Projetos) [20] que a sucesso no desenvolvimento de projetos de e-Commerce depende dos seguintes fatores:

- Ciclos de vida curtos para os projetos;
- Abordagem "Preparar, Apontar, Fogo";
- Poucos recursos capacitados;
- Necessidade de velocidade;

- Mudanças de escopo - no negócio, na tecnologia, nos concorrentes;
- Projetos de alto risco;
- Interdependência entre projetos e conseqüente dificuldade no gerenciamento;
- Processo dirigido pelo negócio e pelos clientes;
- Desenvolvimento de solução para o que for necessário, com a tecnologia existente.

O gerente de projetos de TI par a fazer frente ao novo e arriscado mundo do eCommerce necessita pensar nos fatores que determinam o desenvolvimento dos sistemas e desenvolver um modelo adequado a suas necessidades, conduzindo a flexibilidade e a padronização a um ponto ótimo que um não prejudique o outro. Para o tal, alguns tópicos devem ser levados em consideração:

- Lembre que o cacife do jogo aumentou (ficou mais fácil perder o emprego);
- Conheça o negócio de seus clientes;
- Pense na jornada, não apenas no destino imediato, mas além dele;
- Pense além do dia da implantação (é quando seus problemas aparecerão);
- Estilo e metodologia são importantes (não abra mão);
- Colaboração é fundamental (pense "com" o cliente e não "para");
- A velocidade é mais importante do que a perfeição;
- Cada cliente é o cara mais exigente do mundo;
- Escreva tudo o que fizer (mantenha o controle).

Como conseqüência direta desta nova abordagem de desenvolvimento, os projetos terão tamanhos e prazos menores e atenderão problemas específicos de negócios.

13. Estimativa em Ambiente Cliente / Servidor:

Para que se possa entender melhor o que uma aplicação cliente/servidor, é necessário ficar claro que um aplicativo Cliente/Servidor é constituído por componentes individuais e autônomos que funcionam cooperativamente, de modo a atender, conjuntamente, os requisitos de negócio expressos pelo usuário. Utilizando um protocolo de comunicação padrão, o Cliente solicita serviços e o Servidor os proporciona aos Clientes. Portanto, um aplicativo Cliente/Servidor é constituído pelos seguintes componentes:

- Gerenciamento de Dados: armazenamento e recuperação de dados, de acordo com o definido pelas regras de negócio. Por exemplo, para efetuar uma atualização nos dados dos empregados, o componente de gerenciamento de dados cuida do armazenamento dos dados do empregado.
- Aplicativo: interação entre o gerenciamento de dados e a apresentação, de acordo com o definido pelas regras de negócio. Por exemplo, para executar uma atualização de dados de empregado, o componente aplicativo cuida da validação dos dados do empregado.
- Apresentação: interação com o ambiente, de acordo com o definido pelas regras de negócio. Por exemplo, para executar uma atualização nos dados do empregado, o componente de apresentação cuida da entrada dos dados do empregado.

O usuário de um aplicativo é definido como a pessoa que entende o propósito de negócio do aplicativo e a relação negocial do aplicativo com os outros aplicativos, assim como a interface do aplicativo com o domínio do usuário. O usuário também pode ser qualquer outra pessoa ou coisa que se comunique ou interaja com o aplicativo a qualquer momento.

Voltando para o conceito de Análise de Ponto de Função, as primeiras atividades a serem realizadas em um processo de contagem são: estabelecimento das fronteiras do sistema, como identificar as funções de dados e as funções transacionais (nos termos da APF) dentro do escopo da contagem.

Fato é que a contagem de ponto de função não ajustada só considera as características lógicas dos sistemas, não levando em consideração os fatores técnicos que definiram como serão fisicamente implementadas as funções.

Em uma sistemas cliente / servidor as funções de gerenciamento de dados e os componentes do aplicativo residem completamente no Servidor. Os componentes da apresentação residem inteiramente (remota) ou parcialmente (distribuída) no Cliente, mas são apresentados ao usuário como um todo.

O componente aplicativo encontra-se dividido entre o Cliente e o Servidor, geralmente por razões técnicas. Todas as partes do componente aplicativo devem funcionar como um todo, a fim de atender as definições das regras de negócio.

Os componentes da apresentação residem no Cliente e os componentes de gerenciamento de dados residem no Servidor.

Os componentes do aplicativo e da apresentação residem totalmente no Cliente. No gerenciamento remoto de dados, o componente de gerenciamento de dados reside inteiramente no Servidor. No caso de base de dados distribuída, o componente de gerenciamento de dados é dividido entre o Servidor e o Cliente.

Ao determinar a fronteira do aplicativo, especialmente ao lidar com novas tecnologias, é importante ver o aplicativo do ponto de vista da solução do negócio, ao invés da solução técnica. A solução técnica não deve ter efeito sobre a fronteira do aplicativo. Protocolos e passagem de mensagens entre múltiplas camadas em uma solução técnica multi-camada não afetam a visão que o usuário tem do negócio que o aplicativo pretende suportar. Não é necessário que todos os componentes de um aplicativo residam na mesma plataforma de hardware (isto é, Cliente ou Servidor).

Do ponto de vista do negócio, a fronteira de um aplicativo Cliente/Servidor consiste de todos os componentes que atendem o requisito do negócio, independentemente da implementação física. Concentre-se exclusivamente nos aspectos negociais do aplicativo ao determinar a fronteira do mesmo.

Se o propósito do negócio for o desenvolvimento e a comercialização dos componentes técnicos utilizados no ambiente Cliente/Servidor, então para efeito de medição interna o negócio pode:

- Medir o componente técnico como um aplicativo, e
- Considerar outros componentes técnicos como usuários do aplicativo.

Ao contar os componentes separados do aplicativo Cliente/Servidor, é importante reconhecer que a funcionalidade provida foi incorporada ao aplicativo do usuário final e que essa capacidade será contada por ocasião da contagem desse aplicativo. Dessa maneira, os resultados da contagem do componente separado não serão somados à contagem do aplicativo do usuário, pois estaríamos inflacionando a funcionalidade entregue, com base em uma implementação técnica. Deve ser definida uma carteira de aplicativos independente, especificamente para os componentes Cliente/Servidor.

O objetivo da contagem de pontos de função pode ser determinar o número de pontos de função para um ou mais componentes do aplicativo C/S.

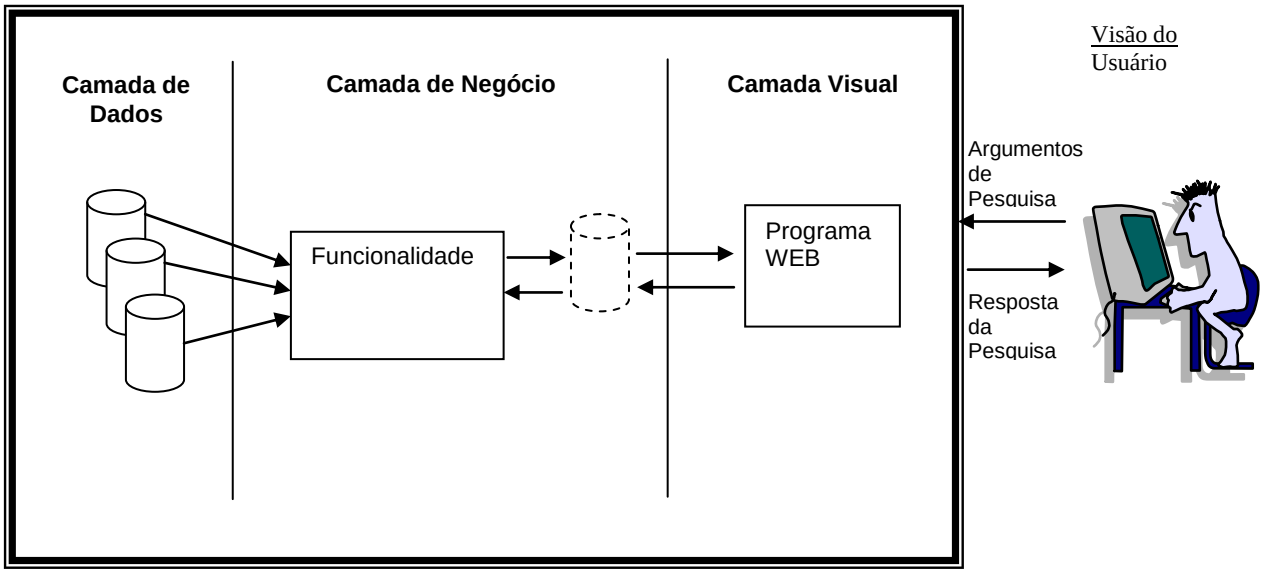
No caso em que todos os componentes do aplicativo C/S estejam no escopo da contagem, o escopo da contagem será dado pela fronteira do aplicativo, se não existirem outras funções (por exemplo, conversão), melhorias ou mudanças pertinentes ao escopo da contagem. Caso um ou mais componentes - mas não todos - estejam no escopo da contagem (por exemplo, se a intenção da contagem for o desenvolvimento de um pacote de gerenciamento de dados), somente as funções que estiverem dentro deste escopo específico deverão ser contadas.

Uma vez que a fronteira da aplicação e o escopo da contagem tenham sido determinados, as regras definidas na APF podem ser aplicadas para identificar as funções de dados e as transacionais.

Por último, as mensagens que passam entre o cliente e o servidor não devem ser contadas como funções separadas. Tais mensagens são consideradas parte da solução técnica.

14. Produtividade e a Qualidade Aplicada nas Estimativas:

No processo de estimativa a partir da visão do usuário, é fator crítico de sucesso a correta definição do que é uma função transacional e o que é parte integrante desta função. Classificar partes como sendo uma função isolada pode causar desvios consideráveis no tamanho do sistema. Sendo assim, se aplicarmos os índices de produtividade sobre o tamanho do sistema, os desvios serão consideravelmente grandes, principalmente pelo fato de que a produtividade média aplicada é resultado de uma série histórica que considera a função transacional como um todo e não como partes[13].



A camada visual, ou interface Web, pode ser vista como a necessidade de implementação de uma camada de interação visual entre o usuário e a camada de negócio. Para o usuário, a interface Web não é vista como uma funcionalidade, pois está totalmente integrada à função transacional completa, como, por exemplo, consultar cliente.

A camada visual não pode ser classificada por si só, segundo os conceitos da métrica análise de pontos de função, como uma das funções transacionais (input, output, query). Portanto, a camada não deve fazer parte da lista de funções

transacionais a serem quantificados para se determinar o tamanho de uma sistema em termos pontos de função.

Ao quantificarmos as características técnicas relacionadas à interface Web, que influencia o fator de ajuste da quantidade de pontos de função, o percentual encontrado referente a estas características não seria significativamente grande a ponto de influenciar no crescimento do tamanho do sistema. Porém, a implementação física desta característica requer um esforço em termos de horas significantemente grande para a implementação desta camada visual.

Um padrão de métrica pode ser considerada aplicável em termos de estimativa de horas quando produz um desvio da ordem de 20% entre o estimado e o realizado.

Esforço em horas para a implementação física de funções

A métrica análise de ponto de função é destinada a quantificar o tamanho de um sistema e não para estimar as horas necessárias para o desenvolvimento. Para tal, um outro fator torna-se necessário ser conhecido: quanto tempo é preciso para implementar um determinado número de pontos de função. Isto é, qual é o índice de produtividade.

A aplicação de índices de produtividade ao tamanho de um sistema, expresso em número de ponto de função, não faz parte do escopo da métrica análise de pontos de função. Ao aplicarmos índices de produtividade para a implementação física das funções quantificadas pela métrica, passamos a tratar o sistema com uma visão influenciada pelas tecnologias a serem utilizadas.

Se forem utilizados dos mesmos conceitos da análise de pontos de função, poderíamos dizer que camada visual pode ser vista como uma “consulta simples”, que recupera dados de um único arquivo lógico e apresenta ao usuário ou que recebe dados do usuário e grava em um único arquivo lógico.

Sendo assim, pode-se determinar o número de pontos de função associado à programação Web e, tendo o índice de produtividade específico para este tipo de ambiente, poderíamos determinar o tempo necessário para a implementação.

Poderia-se considerar que cada funcionalidade transacional a ser implementada na camada de negócio (funcional) demandaria a implementação de aproximadamente uma interface Web na camada visual.

A interface Web, quando implementada em uma linguagem de programação diferente da utilizada na camada de negócio (regras de negócio da funcionalidade), requer a aplicação de índices de produtividade distintos. Por exemplo, se a camada de negócio for implementada em COBOL e a camada visual for implementada em uma linguagem específica para ambiente Web, teremos dois índices de produtividade a serem aplicados.

A partir do levantamento dos requisitos de negócio obtidos da visão do usuário do sistema do usuário, a quantificação do sistema é feita com o auxílio da métrica análise de ponto de função.

As seguintes questões podem ser levantadas quando da definição da quantidade de horas necessárias à implementação física de sistemas:

- A aplicação de índice de produtividade não está no escopo desta métrica.
- A visão do usuário do sistema não contempla a implementação física do sistema.
- A aplicação de índices de produtividade é totalmente dependente da metodologia e dos aspectos tecnológicos envolvidos no desenvolvimento de sistemas.
- A obtenção do tempo necessário para a implementação de interface Web requer a aplicação de índices de produtividade específicos.

A questão da definição do tempo para implementação da camada visual (interface Web) pode ser tratada sob os seguintes aspectos, entre outros:

- Aplicação de índice de produtividade específico, que considera a função transacional on-line como um todo, sobre a quantidade de pontos de função obtidos para a função transacional on-line. Isto nos possibilitaria obter o tempo necessário para a implementação física da função transacional.
- Aplicação de índice de produtividade específico para a camada visual sobre a quantidade de pontos de função originalmente obtida para a função transacional on-line correspondente. Isto nos permitiria obter o tempo necessário para a implementação física da camada visual (interface Web).

- É possível considerar que cada função transacional on-line demandaria a implementação de aproximadamente uma interface Web. Conforme o desenho apresentado anteriormente, a camada visual estaria acessando/atualizando um único arquivo lógico gerado/acessado pela camada de negócio. Portanto, poderíamos aproveitar os conceitos da métrica análise de pontos de função e definir que esta interface Web poderia ser classificada como uma "query" de baixa complexidade. Aplicando-se o índice de produtividade específico para a camada visual sobre a quantidade de pontos de função obtidos, chegaríamos ao tempo necessário para a implementação física da camada visual (interface Web).

Para os três casos citados acima, os índices históricos de produtividade devem ter sido coletados e tratados em concordância com a opção escolhida.

Para finalizar, a coleta e aplicação de índice de produtividade requer padronização. Portanto, a aplicação dos conceitos e definições da métrica análise de pontos de função deve seguir um padrão comum de entendimento, em concordância com os critérios estabelecidos pela métrica, para que a classificação e contagem não sejam feitas a partir de interpretação própria.

15. Tabela de Linguagens de Programação:

Muitas pessoas e organizações utilizam a Tabela de Linguagens de Programação da , Software Productivity Research - SPR, como guia para a estimativa de pontos de função a partir da contagem das linhas de código fonte (SLOC) de um sistema.

Conforme os níveis das linguagens aumentam, menos comandos são necessários para codificar um Ponto de Função. Por exemplo, COBOL é uma linguagem de nível 3 e requer cerca de 105 comandos por Ponto de Função.

Os níveis numéricos de várias linguagens oferecem um atalho conveniente para a conversão de tamanho de uma linguagem para outra. Por exemplo, se uma aplicação requer 1000 comandos de COBOL (não comentários), que é do nível 3, então seriam necessários apenas 500 comandos em uma linguagem de nível 6 (tal como NATURAL) e apenas 250 comandos em uma linguagem de nível 12 (tal como OBJECTIVE C). Conforme pode ser vista, o número médio de comandos requerido é proporcional ao nível da linguagem.

A correlação entre o nível de uma linguagem e a produtividade do desenvolvimento não é linear. Para a maioria dos grandes projetos de software, a codificação representa apenas aproximadamente 30 por cento do esforço total de desenvolvimento de sistemas de informação.

Tomando-se como exemplo um programa que foi escrito em uma linguagem cuja produtividade é o dobro da de uma outra qualquer. O esforço de codificação pode ser reduzido em 50 por cento. Todavia, o projeto total pode melhorar em apenas 15 por cento, uma vez que a codificação representava apenas 30 por cento do esforço total.

Mesmo com a suposta pequena redução no esforço de codificação, quando se transportar estes números para uma escala de projeto com todas as atividades a serem executadas, temos uma diferença considerável que deve ser levada em conta quando da seleção de uma linguagem de programação, a economia de prazo e custo pode significar muito.

Outro fator que não pode ser esquecido é o que diz respeito à manutenção da aplicação. É lógico que quanto mais linha de código possui uma aplicação, mais difícil é a compreensão do que está codificado, mais tempo é gasto em sua manutenção, maior será o custo de manutenção e maior será a possibilidade de erros de execução.

A tabela a seguir demonstra como a linguagem de programação pode influenciar nos prazos e nos custos de desenvolvimento de sistemas de informação [www.spr.com/products/function.htm].

Activity	**Assembler Version (10,000 Lines)**	**Fortran Version (3,000 Lines)**	**Difference**
Requirement	2 Months	2 Months	0
Design	3 Months	3 Months	0
Coding	10 Months	3 Months	-7
Integration/Test	5 Months	3 Months	-2
User Documentation	2 Months	2 Months	0
Management/Support	3 Months	2 Months	-1
Total	25 Months	15 Months	-10
Total Costs	$125,000	$75,000	($50,000)
Cost Per Source Line	$12.50	$25.00	$12.50
Lines Per Person Month	400	200	-200
Cost Per F.P.	$4,166.67	$2,500.00	($1,666.67)
F.P. Per Person Month	1.2	2	+ 0.8

Taxas de produtividade econômicas mais acuradas podem ser obtidas através do exame da média mensal das taxas de produção, em Pontos de Função, associadas aos diversos níveis de linguagem.

A Tabela a seguir mostra como o nível da linguagem afeta a produtividade [www.spr.com/products/function.htm].

Language	**Level**	**Average Source Statements Per Function Point**
1032/AF	20	16
1st Generation default	1	320
2nd Generation default	3	107
3rd Generation default	4	80
4th Generation default	16	20

A área de TI de uma empresa terá dois caminhos básicos para montar uma relação entre quantidade de linhas de código e o número de pontos de função: O primeiro seria adotar uma tabela de conversão adquirida do mercado e ir fazendo ajustes até obter um nível de acerto considerado satisfatório ou Selecionar uma quantidade significativa de funções implementadas nas linguagens de programação comumente utilizada na empresa e montar uma tabela própria.

Para o segundo caso, tem-se os seguintes passos a serem seguidos:

- Selecionar uma quantidade significativa de funções implementadas nas linguagens de programação comumente utilizada na empresa;
- Realizar a contagem de linhas de código de cada uma das funções;
- Realizar a contagem de pontos de função de cada uma das funções;
- Montar uma tabela de relacionamento;
- Acompanhar e realizar os ajustes necessários na tabela conforme as aplicações são desenvolvidas e entregues.

16. Contratação com Base em Pontos de Função:

O fato de parte ou todo trabalho de envolvido no desenvolvimento e manutenção de sistemas estar terceirizado não elimina a necessidade de medição, muito pelo contrário. Tanto no corpo de conhecimento em gerência de projetos (PMBOK) do Project Management institute (PMI) quanto no Capability Maturity Model (CMM) do Software Engineering Institute (SEI) a aplicação de métricas é um aspecto determinante no relacionamento entre a empresa que contrata e a empresa contratada.

Uma forma muito popular de contratação de serviços de desenvolvimento de sistemas é aquela denominada, nos Estados Unidos, body shop (loja de corpos). No Brasil utiliza-se a expressão locação de mão-de-obra. Na operação tipo body shop, uma empresa fornecedora aloca um profissional a uma empresa cliente, cobrando por isso um valor mensal. Essa é uma operação com um risco muito baixo para o locador, pois no caso do profissional alocado não atender ao pretendido pelo cliente, o que se faz é simplesmente substituí-lo. O risco dos projetos não é compartilhado pelo locador de mão-de-obra, que não precisa participar do pesadelo que costuma ser o gerenciamento dos projetos de TI. Claro que há soluções intermediárias, nas quais o fornecedor divide o gerenciamento dos resultados com o cliente, mas essas não podem ser consideradas estritamente como body shop. No body shop propriamente dito, não existe o comprometimento do fornecedor de mão-de-obra com os resultados. Por essa razão, é natural que as empresas busquem soluções alternativas para a contratação do desenvolvimento de sistemas.

Para minimizar os problemas citados anteriormente, algumas grandes empresas resolveram partir para a contratação do desenvolvimento de sistemas baseado em métricas de software. Nessas contratações, o objeto produzido através do contrato - programa, sistema, documento, etc. - é quantificado e pago através de alguma medida objetiva, realizada sobre o próprio objeto gerado. Teoricamente essa é uma solução ideal, pois o que está sendo pago é o resultado, ao invés dos recursos ou insumos

utilizados na sua geração. O problema dessa abordagem passa a ser a escolha, a medição e a interpretação da métrica a ser utilizada na quantificação dos serviços contratados.

No caso do desenvolvimento de sistemas, o principal item que se busca medir é o software produzido através do respectivo contrato. Há contratos específicos para o desenvolvimento de um único sistema, bem como outros destinados ao desenvolvimento e manutenção de diversos sistemas, programas e outros artefatos. Este último tipo de contrato é por vezes denominado "guarda-chuva", por abrigar uma grande variedade de serviços.

As medidas mais utilizadas para a medição de software, tanto no mundo acadêmico quanto na indústria, são as linhas de código e os pontos de função. As linhas de código possuem a grande vantagem da medição automática, através de programas de computador. Tal não é possível com os pontos de função, embora algumas ferramentas busquem realizar essa tarefa, com graus variáveis de acerto. Por outro lado, é difícil estimar linhas de código no início de um projeto, quando ainda não se tem definida a arquitetura; além disso, a quantidade de linhas de código varia com a linguagem utilizada e, o que é pior, com o próprio estilo de programação da equipe. Essas dificuldades têm contribuído para a crescente disseminação dos pontos de função, regulamentados e periodicamente atualizados pelo International Function Point Users Group (IFPUG), organização sem fins lucrativos sediada nos Estados Unidos.

Os pontos de função permitem medir a funcionalidade de um sistema independentemente da técnica utilizada em sua implementação. Dessa maneira, os pontos de função são independentes da linguagem de programação e da plataforma utilizada. Funcionam como a medida de um imóvel em metros quadrados: um apartamento de 100 metros quadrados terá os mesmos 100 metros quadrados, esteja em Manaus ou em Porto Alegre, em um bairro da periferia ou na região mais cara, construído com materiais de primeira linha ou com os mais baratos, finamente decorado ou vazio. Igualmente, um sistema com 500 pontos de função terá esse mesmo tamanho, seja ele implementado através de COBOL/CICS e DB2 no

mainframe, em Delphi com Oracle em uma arquitetura de 3 camadas, ou como uma aplicação web utilizando Java, Solaris e Sybase.

Ao medir um aplicativo utilizando pontos de função, os elementos considerados são componentes visíveis e reconhecidos pelo usuário, tais como arquivos lógicos, transações de entrada e de saída e consultas efetuadas. A visão lógica e centrada no ponto de vista do usuário garante a independência com relação à implementação. Na contagem de pontos de função, é necessário exercitar o processo de abstração utilizado para identificar os componentes contáveis no modelo do sistema, conforme descrito pelo usuário ou registrado na documentação existente. A fim de garantir que todos os contadores de pontos de função utilizem o mesmo procedimento padrão, o IFPUG oferece a certificação profissional. Através dela, tanto o próprio profissional quanto as empresas contratantes podem certificar-se de que as contagens estarão sendo efetuadas corretamente.

Uma vez determinado o tamanho funcional do sistema, o próximo passo é dispor de um método para obter o custo do serviço de desenvolvimento a partir do tamanho em pontos de função. Uma saída simples, mas não recomendável, é estabelecer um custo fixo para cada ponto de função gerado, isto é, um preço por ponto de função. Acontece que o custo de um ponto de função vai variar segundo diversos fatores, dificultando o estabelecimento de um preço único por PF. Na analogia anterior, o ponto de função equivaleria ao metro quadrado, enquanto o preço por ponto de função equivaleria ao preço por metro quadrado. Fica fácil ver que não é possível estabelecer um único preço por metro quadrado para todos os tipos de imóveis, independentemente da localização e do acabamento utilizado. Da mesma maneira, não podemos ter um único preço por ponto de função para todos os sistemas, independentemente da plataforma, linguagem, etc. Esses são fatores que irão afetar a produtividade da equipe de desenvolvedores.

O esforço despendido em um serviço de desenvolvimento é o número de horas gasto para realizá-lo, o qual pode ser dado por: E = F x T, onde E é o esforço em horas, F o tamanho em pontos de função e T a taxa de entrega em horas gastas por ponto de função. A taxa de entrega é o inverso da produtividade. Conhecido o número

de horas E, o custo pode ser obtido multiplicando-se E pelo valor unitário da hora: C = E x H, onde C é o custo do serviço, E é o esforço e H é o valor unitário da hora. Vemos então que, além do tamanho do sistema em pontos de função, precisamos conhecer a taxa de entrega e o valor unitário da hora, para que possamos ter o custo. O tamanho em pontos de função pode ser determinado por um contador de pontos de função experiente. O valor da hora é conhecido do mercado, mesmo porque já é intensamente utilizado nos contratos do tipo body shop. Resta conhecer a taxa de entrega, que reflete a produtividade. Quais empresas brasileiras conhecem sua própria produtividade? Parece que poucas.

Não há estatísticas conhecidas para a taxa de entrega (horas por ponto de função) ou para a produtividade (pontos de função por pessoa por mês) das empresas brasileiras que produzem aplicativos voltados para negócios. Poucas empresas mantêm algum tipo de programa de métricas e, quando o fazem, nem todas utilizam uma medida padrão que permita comparações com outras empresas, o principal benefício na utilização dos pontos de função do IFPUG.

Nesse contexto, faz-se necessário o estabelecimento de programas de métricas, que permitam aos clientes e fornecedores conhecerem sua própria produtividade. Por que uma empresa cliente desejaria conhecer sua própria produtividade? Não bastaria conhecer a produtividade das melhores empresas do mundo e exigir esse mesmo nível dos fornecedores? Talvez sim, no caso da aquisição de uma mercadoria, ou de um serviço totalmente independente do ambiente do cliente. Mas tal não é o caso. No desenvolvimento de sistemas, as demoras, dificuldades e indefinições dos usuários poderão afetar fortemente a fase de concepção ou análise; os processos internos e tecnologia utilizados pela área de TI do cliente poderão, por sua vez, impactar bastante a fase de elaboração ou projeto; os procedimentos de aceitação e a forma adotada para o gerenciamento das mudanças de escopo direcionarão os riscos da fase de construção ou programação; finalmente, a localização geográfica e o cronograma dos usuários irão determinar o tempo para a transição ou implantação do sistema. Para que uma empresa possa exigir, realisticamente, uma determinada produtividade de um fornecedor, é preciso que ela tome como ponto de partida a sua própria produtividade,

buscando melhorias sucessivas ao longo da parceria resultante da contratação. O conhecimento da própria produtividade é, neste caso, o principal objetivo do programa de métricas a ser implantado.

É muito comum as pessoas buscarem números de terceiros, que possam substituir as medições acima indicadas. Este autor recebe, regularmente, uma razoável quantidade de mensagens de colegas que buscam o número mágico. As perguntas mais comuns são do tipo "Em quantas horas se faz um ponto de função, em um ambiente cliente/servidor, usando VB com SQL Server?", ou "Quantos pontos de função faz um programador COBOL por mês?". É claro que tais números existem, nas estatísticas internacionais. O problema é: eles se aplicam ao seu caso? Há uma grande probabilidade de que não.

A variabilidade do processo de desenvolvimento de sistemas é muito grande. Diferentes empresas vão utilizar diferentes metodologias. Muitas empresas não utilizam, consistentemente, qualquer metodologia, embora quase todas disponham de alguma. Quanto uma metodologia padrão é utilizada, os projetos por sua vez são diferentes entre si. Mesmo quando os projetos são comparáveis, muitas vezes as equipes é que não são. Resumindo, o fenômeno que se deseja estudar, o desenvolvimento de sistemas, é tão variável que desafia a definição. Se, ainda assim, aceitarmos que todos que escrevem sobre desenvolvimento de sistemas estão tratando da mesma coisa, teremos que considerar a questão das medições. Para obter a produtividade, é preciso medir o tamanho e o esforço despendido. A medida dos pontos de função pode ser razoavelmente precisa, se forem utilizados contadores experientes. Por outro lado, os critérios para registro do tempo despendido podem variar bastante. Se não houver um padrão para isso, não teremos como saber quem e o quê foi medido. Por exemplo: foi medido o tempo gasto pelos administradores de dados neste projeto? Foi considerado o tempo do pessoal de suporte? O projeto foi medido desde a primeira reunião realizada para tratar do mesmo? O anteprojeto foi considerado nas medições? O treinamento ao usuário foi incluído? Após o aceite final do usuário ainda houve alguma atividade registrada? Deveria ter havido? É mais

importante ter um único critério, consistente, do que ficar discutido se determinado tipo de atividade deve ou não entrar no cômputo das horas despendidas no projeto.

Como conseqüência da grande variabilidade do fenômeno e das respectivas estratégias de medida diferenciadas, as estatísticas internacionais sobre produtividade também variam muito. Isso pode ser verificado através da comparação entre os dados provenientes de três respeitadas fontes de dados sobre as atividades de TI: o International Software Benchmarking Standards Group (ISBSG), Capers Jones e Howard Rubin. Comparando a produtividade do desenvolvimento, medida em pontos de função por pessoa por mês, veremos que o banco de dados do ISBSG registra cerca de 19 PF, Capers Jones indica 7,5 PF e Howard Rubin chega a 3,6 PF. Isso significa uma variação de quase 2 vezes entre Rubin e Jones, e de quase 5 vezes entre Rubin e o ISBSG. Qual desses é o caso da sua empresa? Ou será algum outro? Os dados citados são de 1998.

É inevitável a decepção daqueles que esperam do processo de mensuração de software alguma fórmula mágica ou revelação mística. Exatamente como quando tratamos de metodologias de desenvolvimento, ferramentas CASE ou técnicas de modelagem, não há uma solução brilhante que resolva todos os problemas. Como dizem os americanos, "there is no silver bullet" (não existe a bala de prata). A contratação de software com a utilização de métricas passa, primeiro, pela implantação de um programa de mensuração, também chamado programa de métricas. As estatísticas internacionais são úteis, sim: como ajuda para a validação de nossas próprias métricas e para posicionamento de nosso processo em relação às demais organizações.

A implantação de um programa de métricas requer um esforço específico da organização. O projeto pode ser efetuado em 8 passos, resumidamente descrito a seguir:

1) Documentar o processo de desenvolvimento atualmente utilizado e padronizá-lo (sem buscar a realização de melhorias, mas tão somente a estabilização do processo);

2) Estabelecer os objetivos do programa de métricas (em nosso caso, determinar a produtividade);

3) Definir as métricas necessárias ao alcance dos objetivos pretendidos (por exemplo: tamanho funcional em pontos de função, esforço em horas, qualidade em densidade de erros, etc.);

4) Identificar os dados a serem coletados (por exemplo: horas trabalhadas de cada técnico envolvido, tamanho funcional de cada uma das solicitações de alteração do sistema, erros identificados e suas categorias, etc.);

5) Definir o processo de coleta de dados (identificar os pontos de coleta, periodicidades, formulários ou sistema de coleta, etc.);

6) Obter ferramentas (construir e/ou adquirir as ferramentas que permitam implementar o processo de coleta anteriormente definido);

7) Criar um banco de dados de métricas (implementar um repositório que possa abrigar, de forma organizada e acessível, todos os dados que virão a ser coletados);

8) Definir um mecanismo de feedback (uma maneira que permita à equipe de métricas receber feedback de todos os participantes do processo). Existem várias outras formas de se conduzir o processo de implantação de um programa deste tipo. O importante é o comprometimento da gerência e a clara definição dos objetivos do programa, de modo a garantir que as métricas geradas sejam aquelas necessárias e efetivamente utilizadas.

Uma vez implantado o programa de métricas, a organização começará a obter dados de produtividade. Em uma organização imatura, ainda no nível 1 da classificação CMM, é provável que a produtividade varie de forma errática - refletindo, é claro, o fenômeno que está sendo medido. Gradativamente, será possível separar os projetos por plataforma, ramo de negócio, localização geográfica, perfil da equipe e outros fatores que estejam influenciando os resultados. O tratamento de categorias separadas tenderá a reduzir a variabilidade. Após a estabilização dos fatores mais influentes, será possível estabelecer uma produtividade média para cada categoria definida. Esse será o ponto de partida para a obtenção de melhorias junto aos

fornecedores. Por exemplo, uma empresa poderia firmar um contrato com um fornecedor, tendo como alvo um aumento de produtividade de 25% no primeiro ano e de 10% no segundo (este é apenas um exemplo, valores reais só podem ser determinados mediante cuidadosa análise por parte do cliente e do fornecedor).

A contratação do desenvolvimento de sistemas com a utilização de pontos de função (ou outra métrica) não pode ser realizada exclusivamente com base em dados de terceiros, ainda que publicados em respeitáveis fontes internacionais. Antes da contratação, e imprescindível a implantação de um programa de métricas, que permita ao cliente estabilizar seu processo de desenvolvimento e conhecer sua própria produtividade. Com base nesse conhecimento, melhorias poderão ser buscadas e obtidas junto ao mercado.

Grande ajuda pode ser conseguida através da troca de experiências com empresas assemelhadas, tanto para contratantes quanto para contratados. As melhores práticas de contratação só podem ser conseguidas com o concurso de ambas as partes, em um clima do tipo "ganha-ganha". Tal ambiente não pode ser obtido durante o calor de uma concorrência ou negociação específica. Com o intuito de dar ensejo a tal modo de interação, o Brazilian Function Point Users Group (BFPUG) instituiu um Comitê com a missão de buscar as melhores formas de contratação para o desenvolvimento de sistemas, adequadas aos grandes contratantes e exeqüíveis para os grandes contratados.

CONCLUSÃO

No esforço de implantação de uma nova mentalidade voltada para a qualidade e produtividade, não é admissível o desenvolvimento de sistemas através de ‘feeling’, onde questões como: qual é a produtividade da área de informática, qual é a capacidade de produção, qual conjunto de ferramentas possibilita a maior produtividade, quais são os indicadores de qualidade existentes, qual caminho devo adotar, desenvolver ou comprar um pacote e customizá-lo, são simplesmente deixadas de lado ou respondidas sem suporte de uma base quantificável.

A implantação de um procedimento de métrica possibilita fornecer informações gerenciais para definição de indicadores que suportarão uma série de atividades essenciais para a gerência eficaz de sistemas de informações como: a análise de tendências; análise de impactos na introdução de novas tecnologias sobre a qualidade e produtividade, que pode auxiliar na decisão sobre quais combinações de elementos de tecnologia garantem melhores resultados; a análise de atributos, que permite a comparação da qualidade e produtividade entre plataformas, metodologias, áreas de aplicação, habilidades técnicas de pessoas e assim sucessivamente.

A gerência, criando os padrões de medidas para a instalação, fica em posição de monitorar o comportamento dos projetos e produtos individualmente, analisar resultados, compará-los e verificar a adequabilidade dos respectivos processos e necessidades de implementar melhorias.

A gestão eficiente dos requisitos é um dos principais fatores de sucesso no desenvolvimento de sistemas de informação. Como é dito popularmente “lixo que entra é lixo que sai”, não existe uma fórmula mágica que transforma uma especificação de má qualidade em uma especificação de excelente qualidade.

Ã gestão dos requisitos deve possuir mecanismos que permitem identificar se todos os requisitos necessários ao negócio foram levantados e mapeados. Isto é, se todos os requisitos que estão dentro da fronteira do sistema foram identificados e se está correta a real fronteira do sistema. A simples desconsideração ou esquecimento de

um requisito pode até tornar o projeto inviável. No livro “Gestão de Sonhos” de Amir Klink [33] é citado um episódio sobre o esquecimento de um item de custo extremamente baixo e aparentemente sem importância, que poderia ter inviabilizado a viajem de 1 ano pela Antártica: “Quando estava para deixar o cais em direção ao continente gelado sem escalas, um pessoa no cais me pediu fósforos para acender o cigarro e descobri que não tinha caixa de fósforo a bordo...”.

A implantação de um programa de métricas requer um esforço específico da organização. Requer um compromisso muito forte em enfrentar todos as dificuldades que surgem durante esta longa trajetória sem fim. Não existe espaço para acomodação, sempre está se renovando e apreendendo com as novas informações que são coletadas.

A implantação de um projeto de métricas voltada para a qualidade, pode ser efetuada em 8 passos, como descrito resumidamente a seguir:

1) Documentar o processo de desenvolvimento atualmente utilizado e padronizá-lo (sem buscar a realização de melhorias, mas tão somente a estabilização do processo);

2) Estabelecer os objetivos do programa de métricas (em nosso caso, determinar a produtividade);

3) Definir as métricas necessárias ao alcance dos objetivos pretendidos (por exemplo: tamanho funcional em pontos de função, esforço em horas, qualidade em densidade de erros, etc.);

4) Identificar os dados a serem coletados (por exemplo: horas trabalhadas de cada técnico envolvido, tamanho funcional de cada uma das solicitações de alteração do sistema, erros identificados e suas categorias, etc.);

5) Definir o processo de coleta de dados (identificar os pontos de coleta, periodicidades, formulários ou sistema de coleta, etc.);

6) Obter ferramentas (construir e/ou adquirir as ferramentas que permitam implementar o processo de coleta anteriormente definido);

7) Criar um banco de dados de métricas (implementar um repositório que possa abrigar, de forma organizada e acessível, todos os dados que virão a ser coletados);

8) Definir um mecanismo de feedback (uma maneira que permita à equipe de métricas receber feedback de todos os participantes do processo). Existem várias outras formas de se conduzir o processo de implantação de um programa deste tipo. O importante é o comprometimento da gerência e a clara definição dos objetivos do programa, de modo a garantir que as métricas geradas sejam aquelas necessárias e efetivamente utilizadas.

Uma vez implantado o programa de métricas, a organização começará a obter dados de produtividade. Em uma organização imatura, ainda no nível 1 da classificação CMM, é provável que a produtividade varie de forma errática - refletindo, é claro, o fenômeno que está sendo medido. Gradativamente, será possível separar os projetos por plataforma, ramo de negócio, localização geográfica, perfil da equipe e outros fatores que estejam influenciando os resultados. O tratamento de categorias separadas tenderá a reduzir a variabilidade. Após a estabilização dos fatores mais influentes, será possível estabelecer uma produtividade média para cada categoria definida. Esse será o ponto de partida para a obtenção de melhorias junto aos fornecedores.

REFERÊNCIA BIBLIOGRÁFICA:

1. Sistemática Métrica, Qualidade e produtividade: Carlos Simões - Artigo publicado na Developers' Magazine, setembro de 1999.
2. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 729 1983.
3. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12 1990.
4. Engineering an Effective Measurement Program Course Notes, 1994.
5. Gerência da Melhoria do Processo de Software através de Indicadores da Qualidade e Produtividade - Claudia Hazan.
6. Customizando o Modelo SW-CMM - por Rakesh Shrivastava e Dr. Gular H. Malkani - publicado em Software Quality, julho de 2000.
7. Pontos de Função e o SEI-CMM - David Lipton - Q/P Management Group.
8. NASSCOM - http://www.nasscom.org/, "IT Quality", Dez. 1999.
9. Linstone, Harold A. e Murray Turoff (1975), eds. Delphi Method: Techniques and Applications, Reading, MA: Addison-Wesley Publishing Company, USA.
10. A Tabela de Linguagens de Programação da SPR - T. Capers Jones, Software Productivity Research.
11. NASSCOM - http://www.nasscom.org/, "IT Quality", Dez. 1999.
12. Linstone, Harold A. e Murray Turoff (1975), eds. Delphi Method: Techniques and Applications, Reading, MA: Addison-Wesley Publishing Company, USA.
13. Análise de Pontos de Função e Índices de Produtividade – Carlos Simões – Artigo publicado no BFPUG em 2003.
14. Medida, Métrica ou Indicador: Qual a Diferença? - Bryce Ragland, Software Technology Support Center.
15. Engineering an Effective Measurement Program Course Notes, 1994.
16. Planejamento, Especificação e Execução dos Testes – Carlos Simões – Developers' Magazine 1999
17. IEEE Software Requeriments Specification (Especificação de requisitos de Software (SRS)): IEEE 130.

18. Impacto das novas tecnologias na Contagem de Pontos de Função - Mauricio Aguiar.
19. A Mágica do Gerenciamento de Projetos de eCommerce - Mauricio Aguiar, CFPS - publicado na Developer's Magazine, junho de 2000.
20. e-PM Paradox - Jack Duggal, - Inacom Corporation e University of Hartford.
21. Jones, Capers T. - Estimating Software Costs - McGraw-Hill, 1998.
22. Rubin Systems, Inc. - IT Performance Trends '99 - Meta Group, 1999.
23. International Software Benchmarking Standards Group - The Benchmark Release 5 - ISBSG, 1998.
24. http://www.spr.com/ - Software Productivity Research - Empresa de Capers Jones.
25. http://www.hrubin.com/ - Rubin Systems, Inc. - Empresa de Howard Rubin.
26. http://www.ifpug.org/ - International Function Point Users Group - Site do IFPUG.
27. http://www.bfpug.com.br/ - Brazilian Function Point Users Group.
28. Contratando o Desenvolvimento com Base em Métricas – Artigo de Mauricio Aguiar.
29. Brasil Terra a Vista – A Aventura do Descobrimento – Eduardo Bueno.
30. Projeto & Engenharia de Software – Teste de Software – Trayahú R. Moreira Filho & Emerson Rios.
31. Análise de Pontos de Função – Carlos Eduardo Vazquez, Guilherme Siqueira Simões, Renato machado Albert.
32. Gerência de Projetos de Sistemas – Aguinaldo Aragon Fernandes & José Luiz Carlos Kugler.
33. Gestão de Sonhos - Amir Klink.

ANEXO