UNIVERSIDADE ESTADUAL DE LONDRINA

BRUNO OMENA MESQUITA

# PROPOSTA DE UM MODELO DE GESTÃO DE ESTIMATIVAS DE SOFTWARE APLICADO A FÁBRICA DE TIC GAIA

**LONDRINA - PR**
**2011**

BRUNO OMENA MESQUITA

# PROPOSTA DE UM MODELO DE GESTÃO DE ESTIMATIVAS DE SOFTWARE APLICADO A FÁBRICA DE TIC GAIA

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Rodolfo Miranda de Barros

**LONDRINA - PR**
**2011**

BRUNO OMENA MESQUITA

# PROPOSTA DE UM MODELO DE GESTÃO DE ESTIMATIVAS DE SOFTWARE APLICADO A FÁBRICA DE TIC GAIA

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação da Universidade Estadual de Londrina, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

_____

Prof. Dr. Rodolfo Miranda de Barros
Universidade Estadual de Londrina

_____

Prof. Dr. Jacques Duílio Brancher
Universidade Estadual de Londrina

_____

Prof.ª Ms. Helen Cristina de Mattos Senefonte
Universidade Estadual de Londrina

Londrina, _____de _____de _____.

*A minha família pelo apoio incondicional nesses anos.*

## AGRADECIMENTOS

Agradeço primeiramente a Deus, pela constante proteção e iluminação de meu caminho.

Aos meus pais e irmãos, que com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida, colaborando com tudo que precisei e acima de tudo com muito afeto.

Ao professor e meu orientador Rodolfo Miranda de Barros pelo seu apoio, inspiração e companheirismo, que me ajudou no amadurecimento dos meus conhecimentos e conceitos que me levaram a execução e conclusão desta monografia.

A todos meus amigos (os que fiz em Londrina e os de Penápolis) que sempre estiveram presentes quando precisei e que pude compartilhar muitos momentos felizes no decorrer desses anos.

Agradeço a meus amigos de sala de aula pelo convívio e ajuda em toda a graduação e em especial aos companheiros do Laboratório GAIA e Riguel que nesse último ano mantivemos um convício quase diário com muita descontração, união e apoio em tudo que foi preciso.

A todos os professores e funcionários do Departamento de Computação, que buscaram nos preparar ao máximo para que tivéssemos uma ótima qualificação acadêmica. Mesmo nos momentos difíceis, sempre existiu alguém em quem pudemos confiar e que trabalhou para nos oferecer o melhor.

E por fim, a Universidade Estadual de Londrina, por oferecer a qualidade de ensino e o curso de Ciência da Computação.

MESQUITA, Bruno Omena. **Proposta de um Modelo de Gestão de Estimativas de Software aplicado a Fábrica de TIC GAIA**. 2011. 97p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2011.

## RESUMO

Com o crescimento atual do desenvolvimento de software pelo mundo nos últimos anos, gerenciar bem e planejar o desenvolvimento se tornarão praticas crucias para se coordenar efetivamente um projeto de software. E o processo de estimativa é a base para se planejar e controlar bem os projetos de software. Então quanto mais precisas forem as estimativas melhor será o planejamento, pois este assumirá menos riscos, em contra partida com estimativas ruins ou imprecisas maior o risco que o planejamento assume podendo acarretar em grandes prejuízos para o projeto ou até mesmo o seu cancelamento. O trabalho a ser desenvolvido será uma visão geral sobre o processo de estimativas desde sua definição passando pelas suas praticas, limitações, metodologias, principais técnicas e modelos. Diante do estudo a ser realizado e objetivando agregar qualidade ao processo de desenvolvimento foi proposto um modelo de gestão das estimativas para a fábrica de TIC GAIA. Espera-se que esse modelo possa servir também de base de estudo para outras organizações.

**Palavras-chave:** Estimativas. Software. Metodologias. Modelo. Calibrar

MESQUITA, Bruno Omena. **Proposal for a Model Management Software Estimates applied to the GAIA ICT Factory**. 2011. 97p. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2011.

## ABSTRACT

With the current growth of the world software development in recent years, plan and manage well development practices became vital for effective coordination of a software project. And the estimation process is the basis for planning and control well the software projects. So the more accurate the estimates are better planning, as this will take less risk in starting against bad or inaccurate estimates greater the risk that planning takes may lead to large losses for the project or until its cancellation. The work to be developed will be an overview of the estimation process since its definition by moving their practices, limitations, methodologies and key techniques and models. Front of the study to be conducted and aiming to add quality to the development process was proposed a management model estimates for the ICT factory GAIA. And that this model can serve as a basis for the study of other organizations.

**Key words:** Estimates. Software. Methodologies. Model. Calibrate.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| TIC | Tecnologias da Informação e Comunicação |
| ICT | *Information and Communication Technology* |
| WBS | *Work Breakdown Structure* |
| LOC | *Lines of Code* |
| PF | Pontos de Função |
| PERT | *Program Evaluation and Review Technique* |
| MRE | *Magnitude of Relative Error* |
| *CBR* | *Case-based Reasoning* |
| COCOMO | *COnstrutctive COst MOdel* |
| SLiM | *Software Lifecycle Management* |
| EAF | *Effort Adjustment Factor* |
| SF | *Scale Factor* |
| KDSI | *Kilo Delivered Source Instruction* |
| SLOC | *Source Lines of Code* |
| FSM | *Functional Size Measurement* |
| IFPUG | *Internatinal Function Point International Users* |
| *ISO* | *Internatinal Organization for Standartization* |
| *APF* | Análise de pontos de função. |
| *ALI* | Arquivos Lógicos Internos |
| AIE | Arquivos de Interface Externa |
| EE | Entradas Externas |
| SE | Saídas Externas |
| CE | Consultas Externas |
| NESMA | *Netherlands Software Metrics Association* |
| UCP | *Use Case Points* |
| UAW | *Unadjusted Actor Weight* |
| UUCW | *Unadjusted Use Case Weight* |
| UUCP | *Unadjusted Use Case Points* |
| TCF | *Technical Complexity Factor* |
| ECF | *Environmet Fomplexity Factor* |
| ISBSG | *International Software Benchmarking Standards Group* |

# SUMÁRIO

# 1 INTRODUÇÃO

O crescimento acelerado na indústria de desenvolvimento de software faz com que diversas empresas busquem melhorias no processo de desenvolvimento através de padrões, normas e métodos. Estas melhorias refletem positivamente nos prazos de entrega, custos e qualidade do software, fazendo com que a empresa que os adote obtenha uma vantagem sobre seus concorrentes e maior chance de sucesso competitivo na indústria de desenvolvimento de software.

Dentre as diversas melhorias de processo almejadas por empresas do setor, pode-se destacar a melhoria no processo de estimativas. Estimar o tempo, esforço e data de entrega de um software é um processo delicado, o qual envolve muita sensibilidade do gerente de projetos e de toda equipe, exigindo um grande conhecimento do projeto e capacidade de adaptação. Este cenário, comum às empresas de desenvolvimento de software, ocasiona um aumento nas responsabilidades do gerente de projetos, que só pode contar com sua experiência profissional como ferramenta.

As métricas e estimativas de software vêm se tornando um dos principais tópicos na Engenharia de Software com a crescente exigência de seus consumidores pela qualidade, rapidez, comodidade e baixo custo de implantação e manutenção de software. É impossível não enxergar tais técnicas como alavanca para um produto de melhor qualidade e com custos adequados. Mas existem ainda muitas barreiras que impedem os profissionais da área de utilizarem tais técnicas, como a falta de treinamento, desconhecimento, descrença em relação às métricas, e etc.

Cada método possui suas peculiaridades, seus pontos positivos e negativos, tendo uso somente em casos específicos e possuindo diversas premissas iniciais, que muitas vezes não são facilmente obtidas. Tendo em vista estas peculiaridades, é fácil compreender que cada método tem seu uso específico, seja por uma equipe de desenvolvimento, seja por gerentes de projeto ou outros membros. Um bom método deve levar em consideração a homogeneidade entre processos de desenvolvimento de software, adaptando-se facilmente a cada um e atendendo suas necessidades. Uma vez que as estimativas do projeto se tornem precisas o suficiente ao ponto de minimizar a preocupação com grandes erros de estimativa, elas podem produzir benefícios adicionais, tais como: melhor visibilidade, melhor qualidade, melhor orçamento, identificação adiantada de riscos, entre outras.

Tendo em vista esse conjunto de benefícios ao desenvolvimento e a constante falta da prática de se estimar nos projetos de software é que motivou a elaboração desse trabalho, e que com o estudo realizado foi proposto um modelo de gestão de estimativas para a fábrica de TIC (Tecnologias da Informação e Comunicação) GAIA do Departamento de Computação da Universidade Estadual de Londrina.

Este trabalho está dividido da seguinte forma: no Capítulo 2 serão apresentados os conceitos e relacionamentos mais relevantes sobre estimativas, no Capítulo 3 apresentadas as principais metodologias usadas para se estimar, no Capítulo 4 os principais modelos paramétricos e algumas de suas características. Já no Capítulo 5 são apresentadas as técnicas mais comumente usadas para estimar o tamanho, esforço e o cronograma em projetos de software. No Capítulo 6 foi proposto um modelo de gestão de estimativas da fábrica GAIA baseado no estudo realizado, e finalmente no Capítulo 7 serão apresentadas às conclusões relativas a este trabalho.

# 2 CONCEITOS FUNDAMENTAIS SOBRE ESTIMATIVAS

Neste capítulo serão abordados alguns conceitos sobre estimativas, que vão da sua definição básica até conceitos sobre o relacionamento das estimativas com o projeto como um todo.

## 2.1 DEFINIÇÃO DE ESTIMATIVA

O Segundo Chemuturi [1] termo estimar pode ser definido como:

Uma antecipação inteligente da quantidade de trabalho que precisa ser realizado e os recursos (humanos, monetários, técnicos e recursos de tempo) necessários para executar um trabalho em uma futura data, usando métodos específicos, em um ambiente definido.

Portanto pode-se definir Estimativa de Software como o processo que estima o tamanho do software, ou seja, o montante (quantidade) de software que precisa ser desenvolvido, o esforço necessário para desenvolvê-lo em termos de pessoas-dias, pessoas-horas ou pessoas-mês para uma dada estimativa de tamanho, o custo deste software e seu cronograma de desenvolvimento, sua duração.

E é nessa linha que o trabalho seguirá, apresentando metodologias e técnicas que ajudam a realizar essa antecipação, bem como a relação da estimativa com outras partes do processo de desenvolvimento de software.

## 2.2 ESTIMATIVAS E PLANOS

Estimativas e planejamento são tópicos relacionados, mas estimativa não é planejamento e planejamento não é estimativa. A estimativa deve ser tratada de forma imparcial, um processo analítico, já o planejamento deve ser tratado de forma objetiva. É perigoso a partir da estimativa querer qualquer resposta particular acerca do projeto, o que se

almeja é a precisão[1] e não buscar um resultado particular. Já o objetivo do planejamento é buscar um resultado particular. Se planeja meios específicos para chegar a um fim específico. Estimativas são a base do planejamento bem como do gerenciamento efetivo até o fim do projeto [2] [3], mas os planos não são o mesmo para as estimativas. Se as estimativas estão muito distantes das metas, os planos deveram reconhecer essa lacuna e considerar um projeto de alto risco. Já se as estimativas estão próximas das metas o plano pode assumir um risco menor.

A linha entre estimativas de projeto e planejamento de projeto é larga. Muitos parâmetros do planejamento necessitam ser estimados, incluindo quanto esforço é preciso alocar para a construção do software, para testes, requisitos, *design*; quantos testadores para cada desenvolvedor; quantas horas de esforço pode-se destinar a um projeto específico por semana ou por mês; qual o risco que o projeto assume; entre outros.

Abaixo segue outras considerações do planejamento que dependem em parte da precisão das estimativas [2]:

- Criação de um cronograma detalhado;
- Identificar o curso crítico do projeto;
- Criação de uma completa *Work Breakdown Structure* (WBS);
- Priorização de funcionalidades para a entrega.
- Marcos do projeto.

O Objetivo das Estimativas neste contexto é dar base para a criação de um plano inicial realista. Para que a partir desse ponto o planejamento e controle do projeto, em vez das estimativas, prevaleçam [2].

Em Suma, o planejamento diz como conduzir o projeto, e as estimativas diz o quanto se planejar para uma determinada tarefa ou função.

[1] *O termo precisão no decorrer do texto faz referência a acurácia da estimativa e não a quantidade de dígitos que o valor estimado tem.*

## 2.3 ESTIMATIVAS COMO PROBABILIDADE

Como três quartos dos projetos de software excedem em suas estimativas, as chances de qualquer projeto estar 100% dentro das estimativas de prazo e orçamento não é grande [2]. Diante disso surge uma pergunta: “se as chances não são de 100%, elas são de quanto?”. Essa questão é um dos pontos centrais em estimativas de software.

Normalmente estimativas de software são representadas simplesmente por números, como “O projeto vai levar 10 semanas para ser concluído”. Esse tipo de abordagem não é a mais recomendada, pois não agrega nenhum indicador de probabilidade associado aos números.

Normalmente estimativas usando só números é uma meta mascarada como uma estimativa. O que pode acarretar em estimativas irrealistas, comprometendo todo o planejamento do projeto.

As estimativas de software precisam reconhecer que os projetos de softwares são cercados de incertezas. Coletivamente essas fontes de incertezas significam que os resultados dos projetos seguem uma distribuição de probabilidade, alguns resultados são mais prováveis outros são menos, e um conjunto no meio da distribuição é o mais esperado. Isso leva a crer que a probabilidade dos resultados segue uma distribuição normal (gaussiana) como na figura 1.

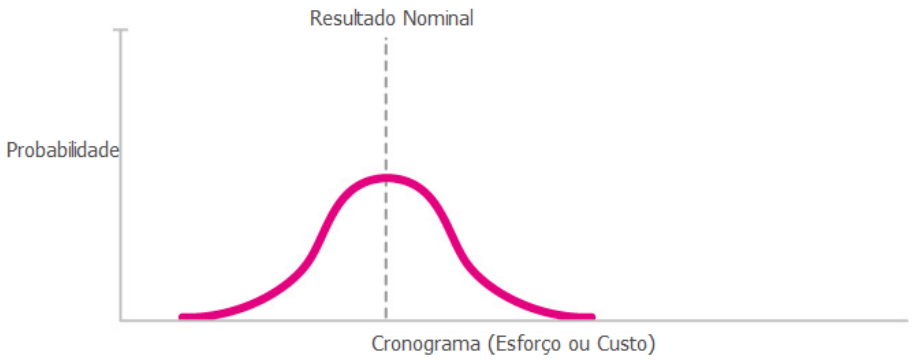


**Figura 1 - Uma suposição comum de que os resultados do projeto de software seguem uma curva de gaussiana. Adaptado de [2].**

Cada ponto da curva representa a chance do projeto acabar exatamente no prazo ou no orçamento. Porém assumir que os resultados são simetricamente distribuídos

sobre uma média não é válido [2]. Há um limite de quão bem um projeto pode ser conduzido, o que significa que o lado esquerdo da distribuição é truncado em vez de estendido como em uma distribuição gaussiana ilustrado na figura 2.

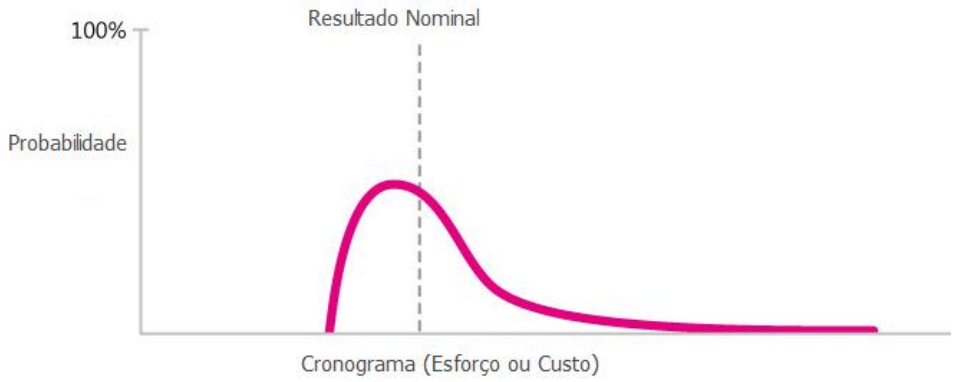


**Figura 2 - Uma descrição precisa dos resultados possíveis do projeto. Adaptado de [2].**

A linha vertical tracejada da figura 2 mostra o resultado nominal, chamado também de “50/50”. Ou seja, há 50% de chance do projeto acabar bem e 50% de chance dele acabar pior. Estatisticamente isso é chamado de resultado médio.

Em vez de expressar a estrega em uma data especifica também é possível expressar a probabilidade de entrega em cada data específica como na figura 3.

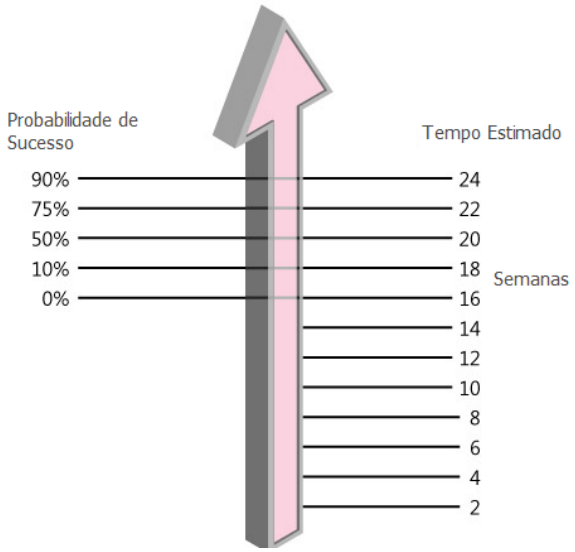


**Figura 3 - Todas as estimativas de valor único são associadas com uma probabilidade, explícita ou implicitamente. Adaptado de [2].**

É possível expressar a probabilidade nas estimativas de várias formas por números. Com um percentual de confiança junto de um valor numérico: “Há 75 % de confiança em 22 semanas”. Pode ser por melhor e pior caso que o implica uma probabilidade: “Melhor caso 18 semanas e pior caso 24 semanas”. Ou simplesmente estabelecer uma faixa de possibilidades: “De 18 a 24 semanas”. O importante é que todas as estimativas incluem probabilidade explicita ou implícita. Estimativas com probabilidade explicita são um sinal de boas estimativas [2].

## 2.4 Definição de Boa Estimativa

Alguns especialistas propuseram várias definições de boas estimativas. Capers Jones [4] afirma que a precisão de ± 10% é possível, porém somente em projetos bem controlados. Em projetos caóticos há muita variabilidade para chegar a esse nível de precisão. Conte, Dunsmore e Shen [5] propuseram que uma boa estimativa deve estar dentro de 25% dos resultados reais e 75% dentro do tempo. Este padrão é o mais aceito e usado para avaliar a precisão das estimativas [6].

## 2.5 Controle do Projeto e Estimativas

Muitas vezes tem-se à impressão que as estimativas de software são feitas por um estimador imparcial, localizado em outro lugar, desconectado do planejamento do projeto.

Na realidade, isso é muito pouco provável. Já que uma vez que são feitas as estimativas, baseando-se nelas são assumidos compromissos de oferecer funcionalidades e qualidade em uma determinada data, então a partir disso o projeto é controlado para atender a essas metas. Tipicamente o controle de projeto inclui atividades como: remover requisitos não cruciais, redefinir requisitos, mudar a equipe do projeto, entre outras. A figura 4 ilustra este dinamismo.

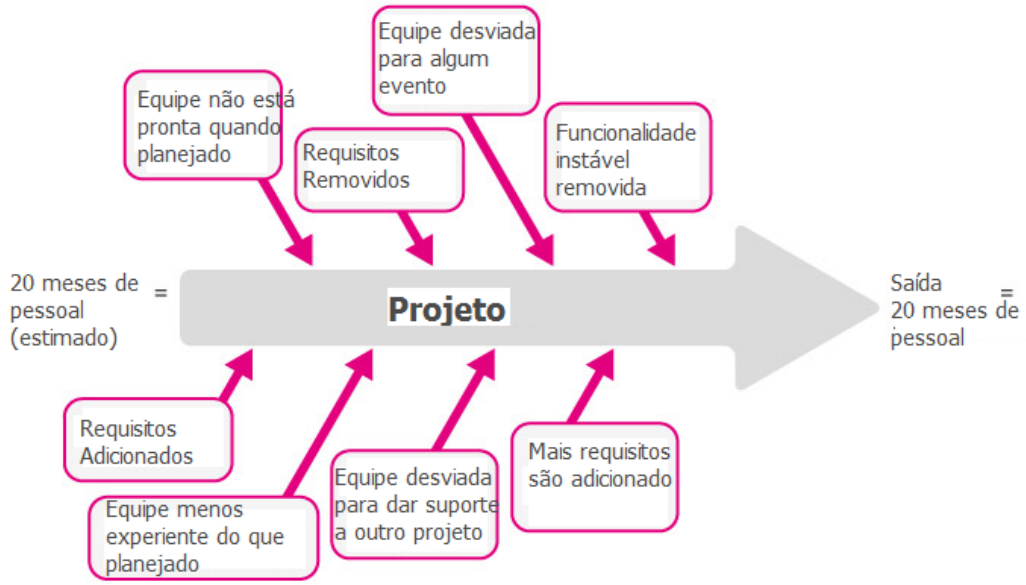


**Figura 4 - Dinamismo das mudanças em um projeto. Adaptado de [2].**

Além das atividades de controle do projeto, este ainda é frequentemente afetado por eventos externos, que são imprevisíveis. Estes eventos que acontecem durante o projeto acabam quase sempre invalidando pressupostos que foram usados para estimar o projeto em primeiro momento. Assim torna-se praticamente impossível fazer uma avaliação clara da precisão das estimativas, pois o projeto que foi entregue não é o mesmo que foi inicialmente estimado.

Dessa forma os critérios para uma boa estimativa não podem ser baseados apenas na capacidade de previsão, que é impossível de avaliar [2]. Mas sim na capacidade de apoiar o sucesso do projeto.

## 2.6 O PROPÓSITO DAS ESTIMATIVAS

Os planos de projetos frequentemente encontram uma lacuna entre as metas e o que é estimado para o orçamento e cronograma. Caso essa lacuna seja pequena, o planejador pode conseguir controlar o projeto levando-o a uma conclusão com sucesso, “apertando” o cronograma, orçamento, ou o conjunto de recursos. Se a lacuna é muito grande as metas do projeto devem ser reconsideradas.

O principal objetivo das estimativas não é prever o resultado de um projeto, é determinar se as metas do projeto são realistas o suficiente para permitir que o projeto seja controlado para atendê-las.

Os problemas começam a aparecer quando a lacuna entre as metas do negócio e as estimativas de esforço e cronograma se tornam muito grandes. Caso as metas e as estimativas divirjam em torno de 20% o gerente de projeto terá base suficiente para controlar o conjunto de recursos, cronograma, tamanho da equipe, e outros parâmetros para atender às metas do negócio [6] [7].

Se a distância entre as metas e o que é realmente necessário for é muito grande, o gerente não vai ser capaz de controlar o projeto para uma conclusão bem sucedida, fazendo pequenos ajustes nos parâmetros do projeto. As metas do projeto terão de ser melhor alinhadas com a realidade antes do gestor poder controlar o projeto para atingir os seus objetivos.

Essa dinâmica mudança de premissas do projeto que causam os maiores erros de estimativas. Precisão de ± 5% não será muito boa caso o os pressupostos do projeto tenham mudado em 100%.

## 2.7 SUPERESTIMAR E SUBESTIMAR

Uma estimativa perfeitamente precisa constitui a base ideal para o planejamento de projetos [3]. Como estimativas precisas são raras [2], é necessário saber onde é melhor errar – subestimando ou superestimando.

### 2.7.1 Contras da Superestimação

Gerentes e outros envolvidos no projeto temem, por vezes, que um projeto seja superestimado e a Lei de Parkinson[2] entrara em ação. Por exemplo, se um desenvolvedor tiver cinco dias para concluir uma tarefa que possa ser concluída em quatro dias, ele vai encontrar alguma maneira de usar o tempo extra. Com isso alguns gerentes conscientemente "apertam" as estimativas para tentar evitar a Lei de Parkinson.

[2] *A idéia da Lei de Parkinson é de que o trabalho será expandido para preencher o tempo disponível.*

Outra preocupação é com a “Síndrome do Estudante” [4]. Se for dado muito tempo aos desenvolvedores, eles vão adiar suas tarefas até o final do projeto, e quando chegar a esse ponto vão correr para conclui-lo, e provavelmente não conseguiram terminar a tempo.

A grande motivação para a subestimação é incluir um senso de urgência na equipe do projeto [1].

### 2.7.2 Contras a Subestimação

A subestimação causa vários problemas, entre eles [2]:

Reduz a eficácia dos planos de projetos. Subestimativas podem alimentar suposições ruins podendo causar erros de planejamento de equipe, como planejar usar uma equipe menor do que necessário. Reduz a capacidade de interação entre grupos que podem não ter seus trabalhos prontos quando deveriam estar.

Se o erro estiver na casa de ± 10%, esses erros não causariam nenhum problema significativo [2]. Mas como mostram vários estudos [4] [9] [10] [11], muitas vezes as estimativas são 100% imprecisas. Quando é assim os planos são praticamente inúteis.

Pouca preocupação com fundamentos técnicos podem levar a resultados ruins. Uma estimativa baixa pode fazer com que não seja gasto tempo suficiente com algumas atividades importantes como engenharia de requisitos e *design.* Se não for dado o foco necessário a atividades como estas, mais tarde no projeto será necessário refazê-las com um custo maior do que se tivesse dado mais ênfase a tais atividades [12] [13]. Isso faz com que o projeto tome mais tempo do que teria tomado com uma estimativa precisa.

A Dinâmica de projeto atrasado pode produzir resultados piores que o normal. Quando um projeto entra em *status* de atrasado, as equipes começam a participar de atividades que não precisariam se engajar caso o projeto estivesse no prazo. O que muitas vezes pode produzir resultados ruins devido à imperícia de membros da equipe em determinadas áreas.

### 2.7.3 Comparando os Contras

A síndrome do estudante pode afetar os projetos de software, mas pode ser tratada pelo rastreamento das tarefas ativas no controle do projeto [2].

Como mostra a figura abaixo os melhores resultados veem com estimativas mais precisas. Se a estimativa for muito baixa, ineficiência no planejamento eleva o custo real e o cronograma do projeto. Caso a estimativa for muito alta, a Lei de Parkinson age como consta na figura 5.

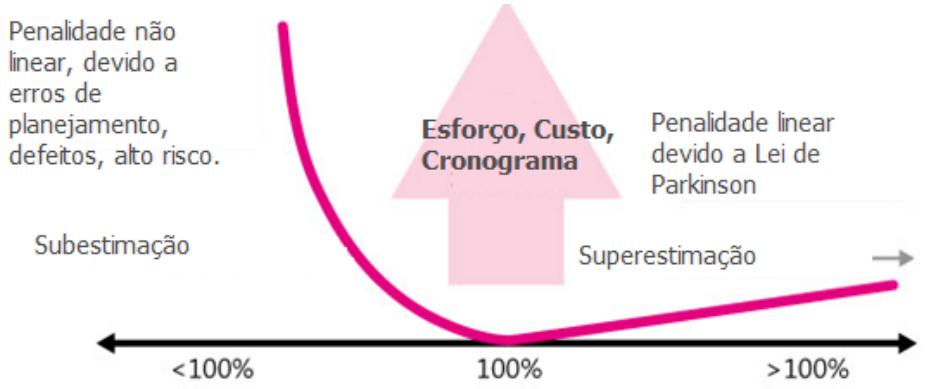


**Figura 5 - Penalidades da superestimação e subestimação. Adaptado de [2].**

Como pode ser observado na figura 5 as penalidades para a subestimação são mais severas do que para a superestimação. Caso não seja possível construir uma estimativa completamente precisa, é melhor errar do lado da superestimação do que da subestimação [2].

## 2.8 Os Erros nas Estimativas

Os erros nas estimativas são em grande parte provenientes de cinco fontes principais [2]:

- Informações inexatas do software que esta sendo estimado;
- Informações imprecisas sobre a capacidade da organização que esta desenvolvendo o software;

- Projeto muito caótico;
- Imprecisões decorrentes do processo de estimativa em si.
- Fontes de incertezas

### 2.8.1 Cone da Incerteza

O desenvolvimento de software consiste em fazer milhares de decisões sobre as questões relacionadas às funcionalidades. A incerteza das estimativas é resultado da incerteza de como essas questões serão resolvidas.

Como resultado desse processo de resolução de decisões, os pesquisadores descobriram que as estimativas de projeto estão sujeitas a uma quantidade previsível de incerteza [2]. O cone da incerteza ilustrado na figura 6 mostra como as estimativas ficam mais precisas conforme o projeto progride.

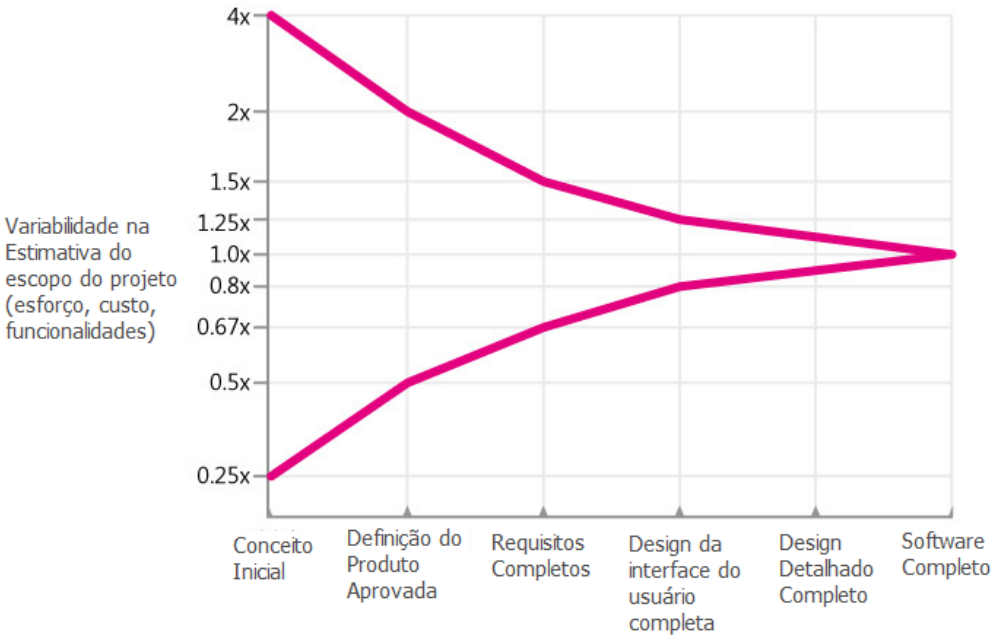


**Figura 6 - Cone da Incerteza, baseado nos marcos mais comuns. Adaptado de [2].**

O eixo horizontal contém os marcos mais comuns em um projeto, como conceito inicial, definição do produto aprovado, requisitos completos e assim por diante.

O eixo vertical contém o grau de erro que foi encontrado nas estimativas.

Como pode ser visto na figura 6 as estimativas criadas no início do projeto estão sujeitas a um alto grau de erro.

Uma questão muito levantada é que se for possível ter mais tempo para estimar é possível produzir estimativas melhores. A resposta para esta questão é não, pois a precisão das estimativas depende do nível de refinamento da descrição do software [2]. Se a definição for mais refinada a estimativa é mais precisa [2].

Uma implicação enganosa do cone da incerteza é de que será necessário muito tempo para o cone estreitar-se, ou seja, uma estimativa boa não estará disponível antes de chegar quase no final do projeto. Isso ocorre porque os marcos são igualmente espaçados na imagem e dá a noção de ser o tempo de calendário.

A figura 7 mostra o cone redesenhado conforme a medida de tempo.

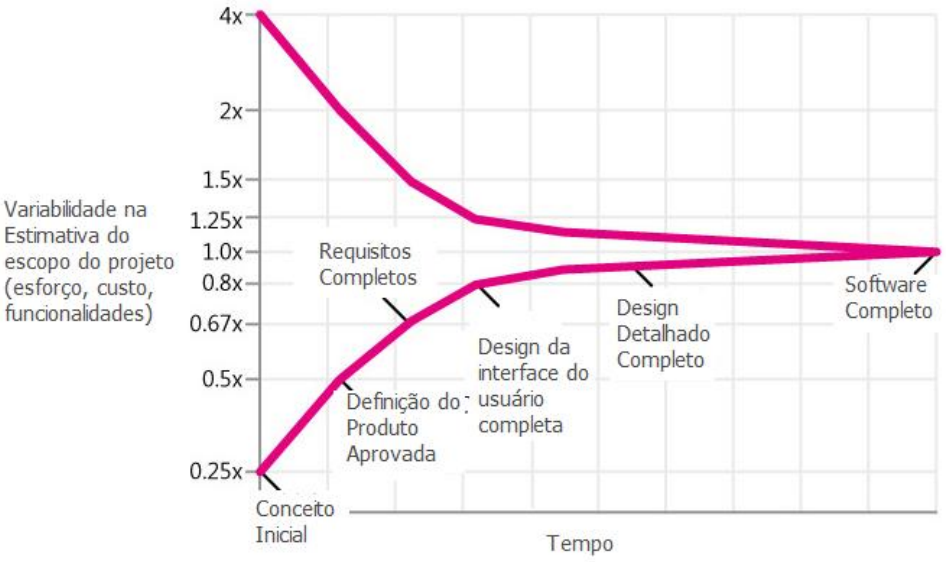


**Figura 7 - Cone da Incerteza baseado no tempo de calendário. Adaptado de [2].**

Observando essa versão do cone, a precisão das estimativas melhoram rapidamente nos primeiros 30% do projeto.

O cone da incerteza representa as estimativas no melhor caso, se o projeto não for bem controlado, ou se os estimadores não são bons as estimativas não irão melhorar. A figura 8 mostra o que acontece quando o projeto não reduz sua variabilidade. A incerteza deixa de ser um cone e passa a ser uma nuvem que persiste até o final do projeto. A questão

não é que as estimativas não convergem é que o projeto em si não converge, isto é, não reduz suficientemente a variabilidade a fim de suportar estimativas mais precisas.

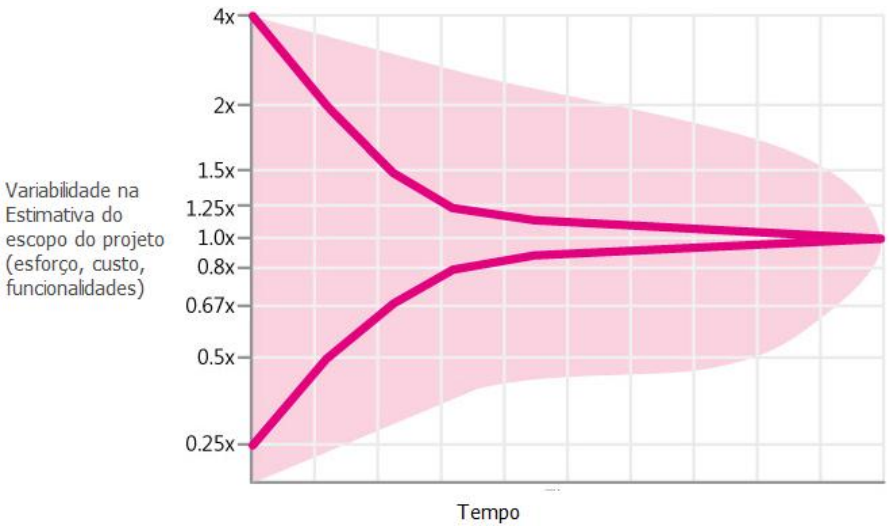


**Figura 8 - Nuvem de incerteza persistindo até o fim do projeto. Adaptado de [2]**

A definição da visão do produto, requisitos, definição da interface do usuário ajudam a reduzir essa variabilidade [4].

### Relação do Cone da Incerteza e Compromissos

As organizações frequentemente sabotam suas estimativas, assumindo compromissos muito cedo no cone da incerteza [2]. Compromissos importantes não são possíveis no início do cone. Organizações efetivas atrasam seus compromissos até que eles tenham feito o cone se estreitar. Compromissos do início para o meio do projeto (± 30% do projeto) são possíveis e aceitáveis [2].

### Processo de Desenvolvimento Caótico

O Cone da Incerteza representa a incerteza que é inerente, mesmo em projetos bem executados. Variabilidade adicional pode surgir de projetos mal executados.

Abaixo alguns exemplos de fontes de caos no projeto [2]:

- Requisitos que não foram investigados muito bem, em primeiro lugar;

- Falta de envolvimento do usuário na validação dos requisitos;
- Desenvolvedores inexperientes levam a muitos erros no código;
- Planejamento incompleto ou não qualificado;
- Abandono do planejado sob pressão;
- Falta de controle automatizado do código fonte.

Estas fontes de caos têm dois pontos em comum. O primeiro é que cada uma apresenta variabilidade que torna difícil estimar com precisão. E o segundo é que a melhor maneira de abordar cada uma destas questões não é através da estimativa, mas através de um melhor controle do projeto.

### 2.8.2 Requisitos Instáveis

A mudança de requisito tem sido frequentemente apontada como uma fonte comum de erro de estimativa [2]. Além dos problemas gerais que os requisitos instáveis causam, estes proporcionam dois específicos desafios à área de estimativas.

O primeiro desafio é que requisitos instáveis representam um contribuinte para um projeto caótico. Se os requisitos não se estabilizarem, o cone da incerteza não irá se afunilar, e a variância nas estimativas vai prevalecer até o final do projeto.

O segundo desafio é que as mudanças de requisitos muitas vezes não são rastreadas e o projeto muitas vezes não é reestimado, quando deveria ser. Em um projeto bem executado, um conjunto inicial de requisitos será a *baseline*, e o custo e cronograma serão estimados a partir dessa *baseline* de requisitos. Como os novos requisitos são adicionados ou antigos requisitos são revisados, as estimativas de custo e cronograma deverão ser alteradas para refletir essas mudanças. Na prática, gerentes de projeto muitas vezes negligenciam a atualizar os seus custos e cronograma quando requisitos mudam [6]. A ironia nestes casos é que a estimativa para a funcionalidade original pode ter sido precisa, mas depois de dezenas de novas exigências que foram acrescentadas no projeto os requisitos que foram acordados antes, não representarão mais o projeto e este não terá nenhuma chance de responder às suas estimativas originais, o projeto será considerado como atrasado, mesmo que todos concordarem que as adições de recursos foram boas ideias.

### 2.8.3 Omissão de Atividades

Uma das fontes mais comuns de erro de estimação é esquecer de incluir tarefas necessárias nas estimativas de projeto [19]. Esse fenômeno se aplica tanto ao nível do planeamento do projeto e de cada desenvolvedor. Um estudo descobriu que os desenvolvedores tendem a estimar com boa precisão o trabalho que se lembrou de estimar, mas eles tendem a ignorar 20% a 30% das tarefas necessárias, o que leva a um erro de estimativa de 20 a 30% [2].

### 2.8.4 Outras fontes de Erros

As fontes de erro descritas são as mais comuns e mais significativas, mas não são as únicas. Aqui estão algumas das outras formas que o erro pode prejudicar uma estimativa [1]:

- Área de negócio desconhecido.
- Tecnologia desconhecida.
- Conversão incorreta de tempo estimado para o tempo do projeto (por exemplo, supondo que a equipe do projeto incidirá sobre o projeto oito horas por dia, cinco dias por semana).
- Processos de orçamentação que minam a eficácia das estimativas (especialmente aqueles que requerem aprovação do orçamento final na parte larga do Cone da Incerteza).

## 2.9 INFLUÊNCIAS NAS ESTIMATIVAS

Influências em um projeto de software podem ser de diversas maneiras. O entendimento dessas influências ajuda a melhorar a acurácia das estimativas e da dinâmica do projeto como um todo.

O tamanho do projeto é a facilmente o mais importante determinante do esforço, orçamento e cronograma. O tipo de software vem em segundo e os fatores pessoais em terceiro lugar. A linguagem de programação e o ambiente de desenvolvimento não são influências de primeira linha para o resultado do projeto, mas para as estimativas sim [2].

### 2.9.1 Tamanho do Projeto

O fator mais importante em uma estimativa é o tamanho do software, pois não há maior variação no tamanho do que em outro fator [2]. A figura 9 mostra a maneira que o esforço cresce em média num projeto comercial, à medida que aumenta tamanho de projeto de 25.000 linhas de código para 1.000.000 linhas de código. A figura 9 expressa o tamanho em linhas de código (LOC – *Lines of Code*), mas a dinâmica seria a mesma se a medida do tamanho fosse em pontos de função (PF), número de requisitos, número de páginas Web, ou qualquer outra medida que expresse a mesma gama de tamanhos.

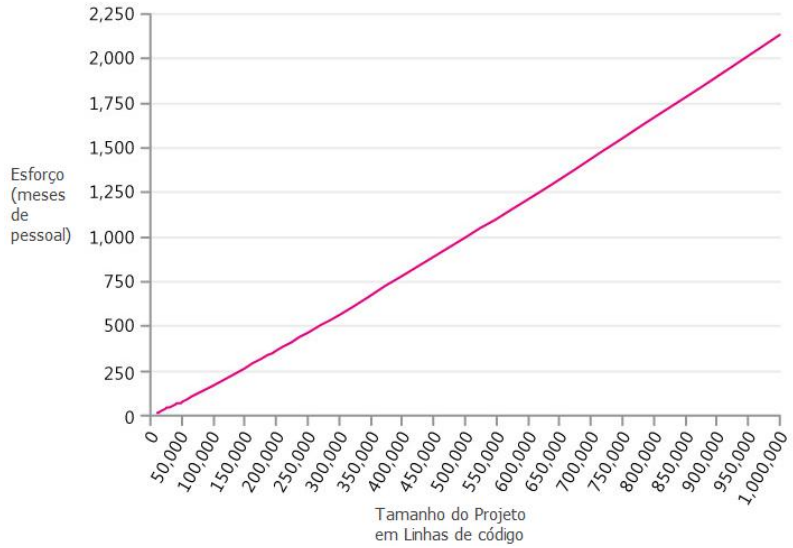


**Figura 9 - Crescimento do esforço em um sistema. Adaptado de [2].**

Como mostra a figura 9, o sistema de 1.000.000 LOC requer muito mais esforço do que um sistema de 100.000 LOC.

Porém normalmente o aumento da dimensão do sistema não obedece a uma função linear como visto na figura 9, na verdade o que ocorre nesse caso é um efeito chamado deseconomia de escala.

**Deseconomia de escala.**

A questão básica é que, em software, projetos maiores exigem coordenação entre grupos maiores de pessoas, o que requer mais comunicação. À medida que aumenta o tamanho do projeto, o número de caminhos de comunicação entre diferentes pessoas aumenta em função do quadrado do número de pessoas no projeto. [2]. A figura 10 ilustra essa dinâmica.

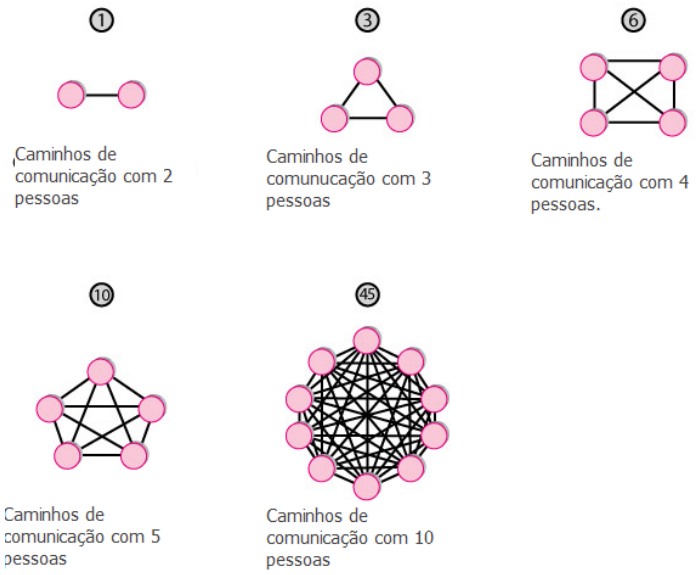


**Figura 10 - Número de caminhos em um projeto crescendo conforme aumentam as pessoas. Adaptado de [2].**

A consequência deste aumento exponencial de caminhos de comunicação (junto com alguns outros fatores) é que os projetos também têm um aumento exponencial do esforço quando aumenta o tamanho do projeto.

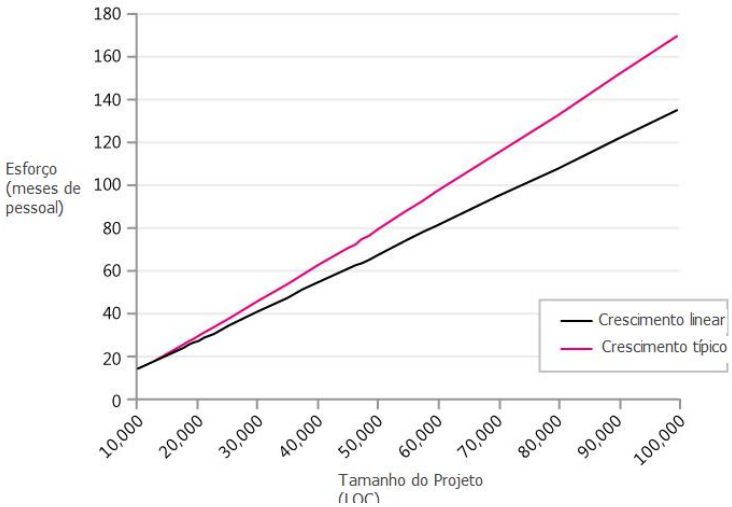


**Figura 11 - Deseconomia de escala em um projeto de negócio. Adaptado de [2].**

Como pode ser visto no na figura 11, o sistema de 10.000 LOC exigiria 13,5 pessoas mês. Se o esforço aumentar de forma linear, um sistema de 100.000 LOC exigiria 135 pessoas mês, mas na verdade exige 170 pessoas mês.

Como ilustrado na figura 11, o efeito da deseconomia de escala não parece muito acentuado. De fato, dentro da faixa de 10.000 a 100.000 LOC, o efeito não é [2]. Mas dois fatores tornam o efeito mais acentuado. Um fator é a maior diferença no tamanho do projeto, e o outro fator é que as condições do projeto se degradam mais rapidamente do que a produtividade média com o aumento do tamanho do projeto.

### 2.9.2 Tipo de Software Que Será Desenvolvido

Depois de tamanho do projeto, o tipo de software que será desenvolvido é a influência maior na estimativa. Por exemplo, uma equipe de desenvolvimento de um sistema de intranet para uso interno pode gerar código de 10 a 20 vezes mais rápido do que uma equipe trabalhando em um projeto de trafego aéreo e em tempo real ou projeto de sistemas embarcados [2].

### 2.9.3 Fatores Pessoais

Fatores pessoais também tem muita influência nos resultados do projeto. De acordo com o Boehm et al. [18] em um projeto de 100.000 LOC os fatores pessoais podem fazer o projeto oscilar em um fator de 22.

Uma implicação destas variações entre os indivíduos é que não se pode estimar com precisão um projeto se não se tem alguma ideia de quem vai fazer o trabalho, porque o desempenho individual varia de acordo com um fator de 10 ou mais.

### 2.9.4 Linguagem de Programação

O uso de uma linguagem de programação específica irá afetar as estimativas pelo menos de quatro maneiras diferentes [2].

Primeiro, a experiência da equipe do projeto com a linguagem específica e ferramentas que serão utilizadas no projeto tem um impacto de ±40% sobre a taxa de produtividade global do projeto.

Segundo, algumas linguagens geram mais funcionalidade por linha de código do que outras.

O terceiro fator relacionado com as linguagens é a riqueza do suporte de ferramentas e ambiente associados com a linguagem. De acordo com Boehm et al [18], um conjunto de ferramentas e ambiente mais fraco irá aumentar o esforço total do projeto em cerca de 50% em comparação com o conjunto mais forte ferramenta e ambiente.

O último fator relacionado com a linguagem de programação é que os desenvolvedores que trabalham em linguagens interpretadas tendem a ser mais produtivos do que aqueles que trabalham em linguagens compiladas, em um fator de 2 [20].

# 3 PRINCIPAIS METODOLOGIAS PARA SE ESTIMAR

Agora serão apresentadas algumas das principais metodologias sobre estimativas de software que podem ser aplicadas a problemas específicos.

Uma importante consideração sobre o uso dessas metodologias é que diferentes técnicas são aplicadas em uma mesma metodologia e diferentes técnicas e metodologias são aplicadas a diferentes problemas.

A técnica ou metodologia mais útil é determinada por dois fatores: pelas influências das estimativas e por evitar as fontes de erros como foi descrito anteriormente.

## 3.1 O QUE SERÁ ESTIMADO

Alguns projetos de software são determinados pelas suas funcionalidades e o foco é determinar o cronograma e o esforço necessário para entregar essas funcionalidades. Outros projetos são determinados pelo seu orçamento e tempo de desenvolvimento então o foco neste caso é estimar quantas funcionalidades podem ser entregues.

Muitas técnicas podem ser aplicadas independentemente do que será estimado, já outras são mais adequadas para estimar quanto de esforço o projeto precisará, quanto tempo ele levará para ser entregue, o orçamento.

As metodologias e técnicas apresentadas nesse trabalho seguem a primeira linha, onde a estimativa é baseada no que é necessário para entregar as funcionalidades.

## 3.2 TAMANHO DO PROJETO

O tamanho do projeto é outro fator a ser considerado para a escolha das técnicas.

**Projetos pequenos:** Segundo MacConell [2] um projeto pequeno é um projeto que envolve cinco ou menos pessoal técnico. Projetos pequenos normalmente não

podem se utilizar de técnicas orientadas a estatísticas como em projetos maiores, pois as variações de produtividade individual inibem os outros fatores, ou seja, a produtividade individual se sobressai.

As melhores técnicas para estimar projetos pequenos costumam ser as baseadas na metodologia *bottom-up* [2], onde normalmente as estimativas são feitas pelos indivíduos que realmente irão fazer o trabalho.

**Projetos grandes:** Um projeto grande é aquele que tem uma equipe de vinte e cinto ou mais pessoas e que dura de seis a doze meses ou mais [2].

As melhores técnicas para estimar projetos grandes mudam significantemente do início ao fim do projeto. No início as melhores técnicas são baseadas na metodologia *top-down*, usando algoritmos e estatísticas [2].

Já no estágio intermediário do projeto a combinação de *bottom-up* e *top-down* baseadas nos dados históricos dos projetos produzem estimativas mais precisas. Nos estágios finais as técnicas "*bottom-up*" produzem estimativas melhores [2].

**Projetos médios:** Projetos considerados médios são os que consistem de cinco a vinte cinco pessoas e duram de três a doze meses. Esses projetos podem usar praticamente todas as técnicas que os projetos grandes usam e muitas das dos projetos pequenos.

## 3.3 CALIBRAÇÃO E DADOS HISTÓRICOS

A calibração é usada para converter contagem em estimativas. Estimativas sempre envolvem alguma espécie de calibração seja ela explicita ou implícita.

Estimativas podem ser calibradas usando qualquer um dos três tipos de dados [2]:

- Dados da Indústria: são dados de outras organizações que desenvolveram basicamente o mesmo tipo de software que esta sendo estimado.
- Dados Históricos: são os dados da própria organização que conduzirá o projeto.

- Dados do projeto: são os dados obtidos nos estágios iniciais do projeto.

Os dados históricos e do projeto são extremamente úteis e podem gerar estimativas de alta precisão [2]. Os dados da indústria são úteis quando os dados históricos e do projeto não estão disponíveis.

Diferentes organizações têm suas próprias características que influenciam seus processos e sua produtividade. Muitas dessas características são difíceis de identificar, muito menos quantificar. Elas irão incluir variáveis como o impacto do ambiente de trabalho, capacidade do pessoal, moralidade da organização, gerenciamento, estrutura da organização, e relacionamento entre cliente/usuário.

Quando o dado é coletado na própria organização, o impacto desses tipos de variáveis é embutido nos dados. A grande dificuldade quando se constrói seu próprio banco de dados históricos é decidir o que coletar.

### 3.3.1 Coleta de Dados

Um bom começo para a coleta de dados é com pequenos conjuntos:

Tamanho: LOC, pontos por função ou qualquer outra coisa que possa ser contada após o software ser lançado.

- Esforço;
- Tempo;
- Defeitos;

Começar com um conjunto pequeno é recomendado pela maioria dos especialistas, para um maior entendimento do que está sendo coletado [1] [4].

A coleta de dados históricos durante o andamento o projeto tende a ser mais fácil.

O principal objetivo da coleta de dados é a conversão desses em um modelo que possa ser usado para fazer estimativas.

## 3.4 JULGAMENTO DE ESPECIALISTAS

O julgamento de especialistas é a forma de estimativa mais usada na prática [17] [22]. Quando se fala em especialista primeiramente deve-se perguntar “especialistas em que?”. Por exemplo, “*expert*” em tecnologia ou em práticas de desenvolvimento não o faz especialista em estimativas. Segundo Jørgensen [17], o aumento da experiência na atividade que esta sendo estimada não leva a uma maior precisão nas estimativas desta atividade. Outro estudo mostra que “*experts*” tendem a usar estratégias simples de estimativas, mesmo quando o nível de experiência no assunto que esta sendo estimado é alta [21] [23].

### 3.4.1 Estimativas de Especialistas Estruturadas

A estimativa feita por especialistas não precisa ser informal ou intuitiva. Como constatado por Lederer and Prasad [24] há diferenças significativas entre as estimativas de especialistas intuitivas e as estruturadas, que muitas vezes produzem estimativas tão precisas quanto às baseadas em modelos formais [17].

#### Melhor e pior caso

O uso do melhor e pior caso é uma boa maneira de melhorar a precisão das estimativas de especialistas. Em vez de estimar cada funcionalidade ou tarefa com um valor único é estimado também quanto de esforço em um possível melhor caso, ou seja, se tudo ocorrer sem problemas e o contrário com o pior caso.

Muitas vezes as estimativas de melhor caso ficam muito próximas das estimativas de valor único comprovando que muitas estimativas são otimistas demais [2].

Esta abordagem tem dois benefícios. Primeiro cria uma consciência de que as estimativas de valor único estão muito próximas do melhor caso. E segundo, o fato de estimar melhor e pior caso começa a criar o hábito de considerar mais o pior caso nas estimativas melhorando assim a gama de resultados possíveis das estimativas de valor único.

Intuitivamente o valor da estimativa seria o valor médio entre o pior e o melhor caso, porém na prática o pior caso é muito ruim comparado ao caso esperado, fato esse que faz o valor médio gerar uma estimativa elevada [2].

Uma técnica chamada *Program Evaluation and Review Technique* (PERT) permite computar o caso esperado que pode não ser exatamente o valor médio entre o melhor e o pior caso [6] [25]. Para usar o PERT é adicionado o caso mais provável no conjunto de casos. Esses por sua vez são estimados pelo especialista. O caso esperado é calculado usando a seguinte fórmula:

**Equação 1**

$$Caso_{esperado} = \frac{[melhor_{caso} + \{4 * caso_{mais\ provável}\} + pior_{caso}]}{6}$$

Esta fórmula é responsável por toda a largura da faixa de possíveis valores, bem como a posição do caso esperado dentro do intervalo.

Há ainda alguns especialistas que sugerem uma alteração na fórmula do PERT, pois existe uma tendência de que as estimativas de caso mais provável sejam otimistas demais [6]. Segue a fórmula alterada:

**Equação 2**

$$Caso_{esperado} = \frac{[melhor_{caso} + \{3 * caso_{mais\ provável}\} + \{2 * pior_{caso}\}]}{6}$$

Segundo MacConnell [2] está é uma solução a curto prazo. A longo prazo deve-se trabalhar com as pessoas para que suas estimativas de caso mais provável sejam mais precisas.

## 3.5 COMPARANDO ESTIMATIVAS

Uma forma de testar a precisão das estimativas é comparar os resultados estimados com os reais. Uma das maneiras mais utilizadas é calcular a magnitude do erro

relativo (MRE - *Magnitude of Relative Error*) [5] [26]. O MRE é calculado usando a seguinte fórmula:

**Equação 3**

$$MRE = [\frac{|Valor_{Atual} - Valor_{Estimado}|}{|Valor_{Atual}|}]$$

Primeiramente o objetivo é que o valor estimado fique dentro do intervalo entre o melhor caso e o pior, posteriormente deve-se diminuir o valor do MRE ao longo do tempo para que as estimativas fiquem cada vez mais precisas.

## *3.6 TOP-DOWN E BOTTOM-UP*

### 3.6.1 *Bottom-up*

*Bottom-up* ou decomposição é pratica de separar uma estimativa em múltiplos pedaços, estimando cada pedaço individualmente, e depois recombina-los em uma estimativa agregada [27].

O método *bottom-up* começa decompondo um projeto em pequenos elementos, que são depois estimados separadamente. Todos esses elementos combinados são frequentemente chamados de Estrutura Analítica do Projeto (WBS - *Work Breakdown Structure*) [27]. Cada elemento pode ser estimado por diversos métodos de estimativas.

**Lei dos grandes números**

A metodologia *bottom-up* é baseada no que os estatísticos chamam de Lei dos grandes números [2]. A essência dessa lei é que se uma estimativa grande é criada a tendência do erro é ser otimista ou pessimista. Mas se são criadas varias estimativas menores, alguns dos erros serão otimistas e outros pessimistas. Assim os erros tendem a se anularem em algum grau. Para começar a se beneficiar dessa lei é recomendado ter de 5 a 10 itens individuais pelo menos [2].

A metodologia *bottom-up* envolve duas etapas: decomposição e integração. Executando esses passos para formar uma estimativa consolidada pode tornar explicito em

muitos sistemas o nível de tarefas, tais como integração, documentação, controle do projeto e gerenciamento de configuração; essas tarefas são geralmente ignoradas em outras abordagens [28].

Em desvantagem essa metodologia requer um extensivo conhecimento não somente do que o software irá fazer, mas também qual papel será atribuído aos membros da equipe e qual a abordagem de gerenciamento será utilizada. Esse tipo de informação provavelmente não estará disponível nos estágios iniciais do projeto. Além disso, o *bottom-up* incorpora a desvantagem do método selecionado para estimar as tarefas individuais.

Esse método deve ser usado por peritos com grande experiência, conhecimento, ou quando os dados do projeto decomposto estão disponíveis [28]. O *bottom-up* leva mais tempo para ser estimado então ele deve ser usado quando há tempo disponível para a realização das estimativas.

### 3.6.2 *Top-down*

De modo similar a metodologia *bottom-up* o *top-down* é feito por decompondo um projeto em elementos de níveis mais baixos ou em fases do clico de vida. No entanto, o esforço estimado para cada elemento é baseado nas características gerais do projeto, vez de características funcionais ou de *design*, isto é no que o software irá fazer a grosso modo [28]. Os elementos individuais são geralmente estimados por analogia ou por julgamento de especialistas. As estimativas individuais são combinadas para desenvolver uma estimativa total do projeto. Embora este método possa parecer mais rápido e mais fácil do que outras abordagens, ele pode ser menos preciso, porque um analista pode facilmente esquecer nível mais baixo de detalhes ou fatores de custo significativo que se tornam visíveis por outras técnicas de estimativa. Além disso, esse detalhe faz a estimativa limitada e difícil de documentar ou verificar, ou para comparar com estimativas mais detalhadas.

Este método pode ser difícil de aplicar no início do ciclo de vida do projeto [28].

Ambas as metodologias podem ser aplicadas na fase de licitação do projeto, planejamento, e reestimativa. Há, no entanto, situação que o uso do *top-down* é mais indicado como em estimativas iniciais com vaga especificação de requisitos onde ainda não é possível construir uma WBS, e situações em que o *bottom-up* é mais indicado como na fase de

reestimativa. Na maioria das situações de estimativas, ambas as estratégias são razoáveis para estimar projetos de software [29].

No estudo realizado por Jørgensen [29] consta que para realizar estimativas utilizando *bottom-up* é necessário um maior entendimento sobre a especificação de requisitos do que a metodologia *top-down*, além de um bom conhecimento dos estimadores de como o software será construído. A estratégia *top-down* pode prover estimativas razoavelmente precisas com um baixo custo e sem muito conhecimento técnico. Porém é necessário usar muito mais as informações de projetos passados, sendo assim para ter sucesso com essa metodologia precisa-se de um bom banco de dados históricos de projetos, e quanto mais similar os dados mais precisa será a estimativa.

Já estratégia *Bottom-up* necessita de maior conhecimento técnico e mais tempo para estimar.

A grande vantagem do *bottom-up* como já citado é que há maior precisão estimando tarefas decompostas do que o projeto todo [29].

Essa diferença favorece o *bottom-up* em situações que é possível forçar os estimadores a gastar o mínimo de tempo e entendimento das implicações da especificação de requisitos, e o *top-down* em situações onde é essencial produzir estimativas com pouco custo e/ou sem gasto de esforço.

## 3.7 ESTIMATIVA POR ANALOGIA

O pensamento por trás da analogia é similar ao raciocínio básico do ser humano usado por quase todos os indivíduos diariamente para resolver problemas espelhados em eventos semelhantes que ocorreram no passado. Analogia é um paradigma muito antigo é fruto de discussões a centenas de anos. Teve um papel muito importante na Grécia antiga, como um método de sugerir ou argumentar sobre fenômenos naturais [29].

Na engenharia de software a analogia utilizar-se de conhecimento específico de experimentos passados para obter soluções de novos problemas ou casos. A solução é derivada encontrando casos similares e usando-os nesses novos problemas [30].

### 3.7.1 Analogia e o Processo CBR

Analogia segue o raciocínio baseado em casos (*CBR - Case-based reasoning),* que é uma técnica que busca resolver novos problemas adaptando soluções utilizadas para resolver problemas anteriores.

O processo CBR é baseado em 4 estágios gerais [30]:

**Recuperação**: a partir da apresentação ao sistema de um novo problema é feita a recuperação na base de casos daquele mais parecido com o problema em questão. Isto é feito a partir da identificação das características mais significantes em comum entre os casos.

**Reutilização**: a partir do caso recuperado é feita a reutilização da solução associada aquele caso. Geralmente a solução do caso recuperado é transferida ao novo problema diretamente como sua solução.

**Revisão**: é feita quando a solução não pode ser aplicada diretamente ao novo problema. O sistema avalia as diferenças entre os problemas (o novo e o recuperado), quais as partes do caso recuperado são semelhantes ao novo caso e podem ser transferidas adaptando assim a solução do caso recuperado da base à solução do novo caso.

**Retenção**: é o processo de armazenar o novo caso e sua respectiva solução para futuras recuperações. O sistema irá decidir qual informação armazenar e de que forma.

A figura 12 descreve o caso geral do Processo CBR, mostrando as etapas importantes e a interação de casa estágio na aplicação do processo.

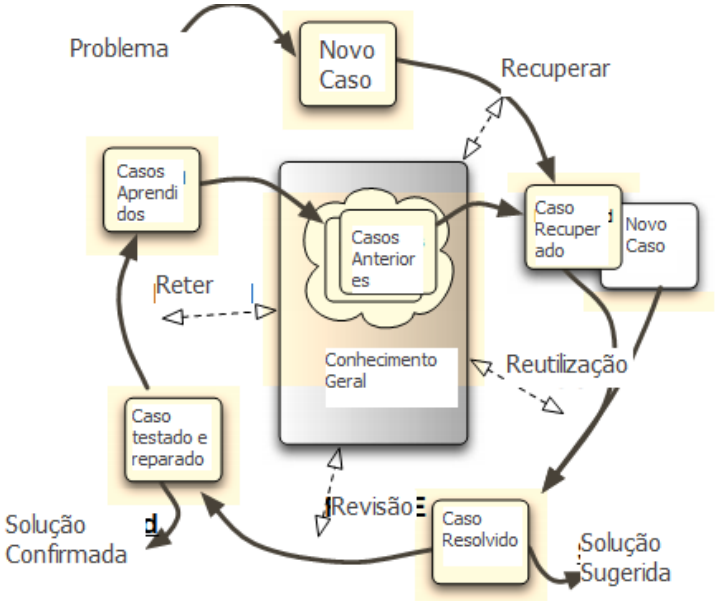


**Figura 12 - Anatomia do ciclo CBR. Adaptado de [30].**

Uma descrição inicial de um problema é um novo caso. Este novo caso é usado para Recuperação de um caso dos casos passados. O caso recuperado é combinado com o novo caso através da Reutilização de um caso resolvido. A solução é testada para o sucesso pelo processo de Revisão. Durante a Retenção, as experiências ou informações mais úteis serão armazenadas para serem usadas no futuro. E o caso base é atualizado para um novo caso apreendido ou um caso modificado.

A semelhança entre o caso alvo e cada caso do CBR é determinada por uma medida de similaridade. A distância euclidiana é a métrica mais comum usada no CBR [30].

### 3.7.2 A Aplicação na Estimativa de Software

Em contraste com o julgamento de especialistas, que não é considerado um método empírico, pois o meio de derivar uma estimativa não é explicito [31], a analogia é mais formal e sistemática do que o julgamento de especialistas usando comparação direta com um ou mais projetos passados. Em muitas circunstâncias, a analogia prove uma boa alternativa a outros métodos algorítmicos principalmente quando não há uma quantidade

suficiente de dados estatísticos disponíveis, porém deve haver pelo menos um projeto com custos e cronograma similares disponível.

A aplicação geral do processo de estimativa por analogia é bem próximo do CBR normal, e envolve o seguinte conjunto de passos [30]:

**Identificação de casos análogos**: O estimador analisa o novo projeto, depois mede ou estima qual serão as funcionalidades desde projeto. Em seguida consulta o repositório de projetos, procura projetos passados com casos similares ao novo projeto e seleciona um ou mais projetos similares como fonte de analogia.

**Previsão do caso alvo**: O valor da estimativa do caso análogo entra como estimativa inicial do projeto alvo. Varias técnicas de adaptação podem ser utilizadas, como uma simples média ou até regressão linear.

**Ajuste final da estimativa:** O ajuste final é aplicado na estimativa inicial para saber a diferença entre as estimativas do projeto análogo e do projeto alvo. O julgamento de especialista é a forma mais comum de ajuste.

Uma vez que o projeto foi estimado, avaliado e revisado, as experiências provavelmente serão úteis para futuros problemas, então este projeto deve ser guardado como um futuro caso de referência.

Cada caso de projeto consiste em um vetor de características do projeto que pode incluir, por exemplo, entradas, saídas, pontos por função, ambiente de desenvolvimento, linguagem de programação entre outros. O vetor de características pode incluir diferentes tipos de dados o que acaba criando alguma complexidade na maneira que a distância entre os casos é mensurada. A distância euclidiana é a melhor métrica para manipular dados contínuos. Um simples ajuste é transformar dados categóricos, como linguagem de programação, em variáveis binárias, e depois normalizar todas as outras variáveis em um valor escalar entre 0 e 1, isso permite que variáveis de ambos os tipos possam ser igualmente comparáveis [32] [33].

Apesar dos muitos benefícios de se utilizar analogia e o fato do conceito de estimativa por analogia ser simples e em muitos casos tem resultados comparáveis aos melhores métodos algorítmicos, há inconvenientes e dificuldades com a estimativa baseada em analogia que modera suas vantagens.

O algoritmo CBR usando na analogia apresenta algumas deficiências como:

- Intolerância a ruídos.
- Intolerância a características irrelevantes.
- Sensível à escolha do algoritmo de similaridade.
- Não há uma maneira simples de processar características categóricas.

Além disso, o algoritmo de busca também tem influência na precisão da estimativa. Walkerden e Jeffery [34] descrevem quatro fatores importantes que influenciam a precisão das estimativas usando analogia, são eles:

- A disponibilidade de um apropriado caso análogo (Conjunto de dados é relevante).
- A solidez da estratégia de seleção (Características do subconjunto de seleção).
- A diferença entre o caso análogo e o caso alvo. (Medida da distância)
- A precisão dos dados usados. (Qualidade do conjunto de dados).
- E caso a seleção seja baseada no julgamento de especialistas a estimativa ficara sujeita as deficiências desse método.

Deve ser considerado também que para as estimativas baseadas em analogia terem sucesso deve-se ter um bom banco de dados histórico.

## 3.8 JULGAMENTO DE ESPECIALISTAS EM GRUPOS

Essa técnica é muito útil para fazer estimativas no início do projeto. E em sistemas desconhecidos.

### 3.8.1 Revisão de Grupos

Uma técnica simples para melhorar a precisão das estimativas criadas por indivíduos é ter um grupo de revisão das estimativas. Quando se tem um grupo de análise de estimativas, esse procedimento requer três regras básicas [2]:

1. Cada membro do grupo estima uma parte do projeto individualmente e depois se reúnem para comparar suas estimativas. Discutir as diferenças nas estimativas é suficiente para entender as fontes de erros. O objetivo é chegar a um consenso sobre os limites superior e inferior das estimativas.
2. Não basta apenas encontrar a média das estimativas. A média pode ser calculada, mas é necessário também discutir as diferenças entre os resultados individuais.
3. Chegar a uma estimativa que todo o grupo aceite.

A melhoria na precisão desta técnica simples é significativa. A figura 13 ilustra os resultados de 24 grupos de estimadores [2]:

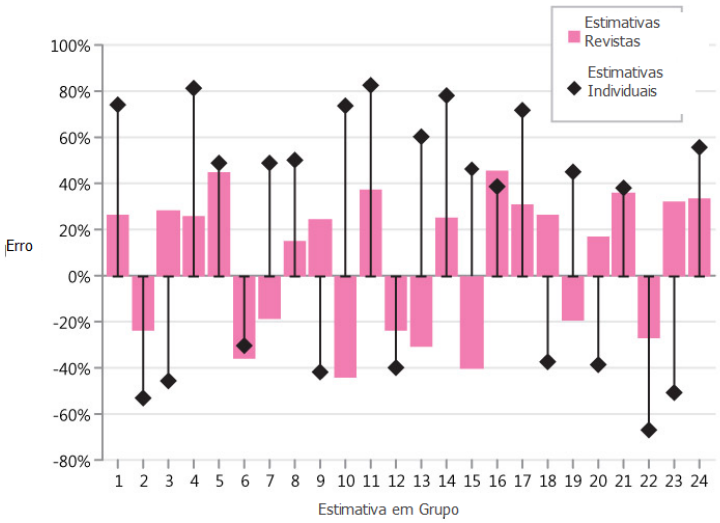


**Figura 13 - Melhora na estimativa, pela revisão em grupos. Adaptado de [2].**

As estimativas individuais na figura 13 o MRE médio é de 55%. Depois da revisão do grupo o erro caiu para 30%. Neste conjunto de estimativas, 92% das estimativas de grupo foram mais precisas do que as estimativas individuais e, em média, as revisões cortam a magnitude do erro aproximadamente pela metade [2].

Estudos em outros campos descobriram que o uso de 3 a 5 especialistas com diferentes formações, funções ou que usam diferentes técnicas parece ser suficiente [17] [35].

### 3.8.2 *Wideband delphi*

*Wideband Delphi* é uma técnica de estimativa de grupo estruturada. A técnica *Delphi* original foi desenvolvida pela *Rand Corporation* no final de 1940 para prever tendências em tecnologia [7]. A técnica *Delphi* básica convoca vários especialistas para criar estimativas independentes e depois de se reunir por quanto tempo for necessário para convergir, ou pelo menos acordar em uma única estimativa. Em estudo inicial sobre o uso do *Delphi* para estimativa de software descobriram que a técnica *Delphi* básica não era mais precisa do que uma reunião de grupo menos estruturada. Barry Boehm e seus colegas concluíram que as reuniões genéricas *Delphi* estavam sujeitas a muita pressão política e também eram susceptíveis de ser dominada pelos estimadores mais assertivos no grupo. Consequentemente, Boehm e seus colegas estenderam a técnica *Delphi* básica em que se tornou conhecida como *Wideband Delphi*. A Tabela 1 descreve o procedimento básico.

| |
|---|
| 1. O coordenador *Delphi* apresenta cada estimador com a especificação e um formulário de estimativa;<br>2. Estimadores preparam as estimativas iniciais individualmente. (Opcionalmente, este passo pode ser pré-formado após a etapa 3.)<br>3. O coordenador convoca uma reunião do grupo em que os estimadores discutem questões de estimativa relacionadas com o projeto em mãos. Se o grupo concordar em uma única estimativa sem muita discussão, o coordenador atribui a alguém para argumentar contra.<br>4. Estimadores entregam suas estimativas individuais para o coordenador de forma anônima.<br>5. O coordenador prepara um resumo das estimativas em um formulário de iteração (figura 14) e apresenta a forma de iteração para os estimadores para que eles possam ver como vão as suas estimativas e comparar com estimativas dos outros.<br>6. O coordenador tem que reunir os estimadores para discutir variações nas suas estimativas.<br>7. Estimadores votam anonimamente em se querem aceitar a estimativa média. Se algum dos estimadores de votar "não", eles retornam para a etapa 3.<br>8. A estimativa final é a estimativa de um único valor, decorrentes do exercício de *Delphi*. Ou, a estimativa final é a faixa criada por meio da discussão *Delphi* e o único valor da estimativa *Delphi* é o caso esperado. |

**Tabela 1 - Técnica *Widebrand Delphi*. Adaptado de [8]**

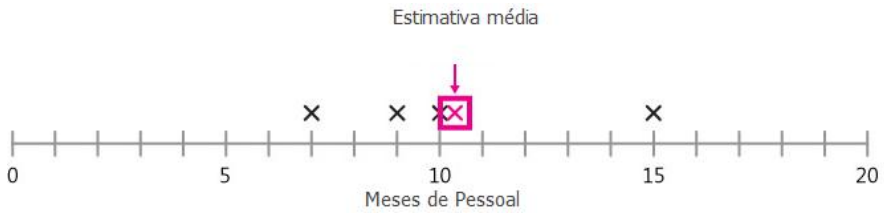


**Figura 14 - Forma de estimativa. Adaptado de [2].**

A forma de estimativa é mostrada na figura 14 que pode ser em forma de papel ou pode ser desenhado pelo coordenador em um quadro. A forma mostra um intervalo de 0 a 20 meses de pessoal. O intervalo que inicialmente é mostrado no formulário deve ser pelo menos o triplo do intervalo esperado pelos estimadores para que os eles não se sintam pressionados por um intervalo predefinido.

É útil para mostrar todos os rounds de estimativas na mesma escala, para que os estimadores possam observar como as suas estimativas estão convergindo (ou, em alguns casos, divergindo).

*Wideband Delphi* é útil se um trabalho esta sendo estimado em uma nova área de negócio, em uma nova tecnologia, ou um novo tipo de software [2].

# 4 MODELOS PARAMÉTRICOS

Nos modelos paramétricos, ou algorítmicos, o esforço, cronograma e custos são estimados usando relações matemáticas, que correlaciona alguns parâmetros como, por exemplo, o tamanho do software, complexidade, capacidade da equipe. Modelos paramétricos têm de ser calibrados com o ambiente de desenvolvimento que são usados [2]. Essa calibragem é feita usando dados históricos e envolve ajustes de pesos de diversos fatores nas fórmulas matemáticas. Os mais famosos exemplos de modelos paramétricos são o *COnstrutctive COst MOdel* (COCOMO) [7] [18], Putman's *SLIM (Software LIfecycle Management*), *Price System's PRICE-S,* e *Galorath's SEER-SEM*. Nessa seção serão apresentadas algumas de suas principais características.

## 4.1 COCOMO E COCOMO II

A versão inicial do COCOMO (também chamado de COCOMO 81) foi desenvolvida pelo Dr. Barry Boehm e publicada no livro *Software Engineering Economics* [7] em 1981. Esse modelo usa regras básicas de regressão linear para prever o esforço, cronograma, custo e pessoal em projetos de software com base nos dados históricos.

Na primeira versão do COCOMO Barry Boehm analisou 63 projetos de software a fim de achar qual a influência do esforço e do tempo de desenvolvimento nesses projetos. Boehm identificou que o tamanho do software é a principal influência do fator esforço. Outros fatores de influência foram o tipo de projeto a ser desenvolvido, as habilidades dos desenvolvedores bem como características de desempenho.

Essa versão do COCOMO é a mais simples e captura apenas algumas relações que influenciam o esforço e o custo. Esse foi o motivo que extensões do modelo básico foram desenvolvidas. O modo intermediário e o modo detalhado do modelo usam um fator de ajuste do esforço (EAF - *Effort adjustment factor*) que é multiplicado pela equação básica para integrar mais características do projeto e produzir estimativas mais precisas. O fator de ajuste é o produto de 15 fatores de custo que podem ser subdivididos em diferentes categorias de acordo com os custos que são relacionados. Essas categorias são custos de plataforma, custos de produto, custos de pessoal e custos de projeto.

Não houve mais nenhuma investigação feita para o COCOMO na sua versão básica, ele está desatualizado e na maioria dos casos é mais aconselhável usar o COCOMO II (COCOMO 2000) [18]. No entanto, os conceitos fundamentais do COCOMO e do COCOMO II são os mesmos.

O modelo inicial do COCOMO (COCOMO 81) tem algumas desvantagens que foram melhoradas no COCOMO II. Os objetivos do COCOMO II são [18]:

- Prover estimativas de custo e cronograma precisas para os projetos atuais e futuros.
- Facilitar a recalibração, adequar e estender o COCOMO II para melhor atender a situações únicas.
- Prover um cuidadoso e fácil entendimento das definições do modelo de entrada, saída e pressupostos.
- Fornecer um modelo construtivo.
- Fornecer um modelo normalizado.
- Fornecer um modelo em evolução.

COCOMO II foi desenvolvido na década de 90 e calibrado usando um conjunto de dados de 161 projetos. Além dos objetivos do modelo, ele foi adaptado para dar conta de [36]:

- Modelos ágeis e não sequenciais de desenvolvimento.
- Abordagens de reutilização envolvendo os pacotes COTS (*Commercial off-the-shelf),* reengenharia, composição de aplicativos e capacidade de geração de aplicativos;
- Abordagem orientada a objetos suportado por middlewares distribuídos;
- Efeitos de processos de maturidade de software e processo de estimativa dirigido à qualidade.

Na equação básica do COCOMO II o esforço em pessoas mês é dado por:

**Equação 4**

$$PM = A * Tam^{E} * \prod_{i=1}^{n} EM_i$$

Onde $A = 2.94$ para o COCOMO II [18].

O fator $A$ é uma constante. O expoente $E$ é um agregado de cinco fatores de escala (SF – *Scale Factor*) que representam a deseconomia de escala encontrada nos projetos de software. A constante $E$ pode ser calibrada ao ambiente. A grande melhoria do COCOMO II é que novos fatores de custo foram adicionados e que o modelo foi calibrado com uma amostra maior e com dados mais recentes. A tabela 2 contém os direcionadores de custos do COCOMO II:

| | |
|---|---|
| Produto<br>- Required Software Reliability (RELY)<br>- Data Base Size (DATA)<br>- Product Complexity (CPLX)<br>- Developed for Reusability (RUSE)<br>- Documentation Match to Lifecycle Needs (DOCU) | Pessoal<br>- Analyst Capability (ACAP)<br>- Programmer Capability (PCAP)<br>- Personnel Continuity (PCON)<br>- Application Experience (APEX)<br>- Platform Experience (PLEX)<br>- Language and Toolset Experience (LTEX) |
| Plataforma<br>- Time Constraint (TIME)<br>- Storage Constraint (STOR)<br>- Platform Volatility (PVOL) | Projeto<br>- Use of Software Tools (TOOL)<br>- Multisite Development (SITE)<br>- Required Development Schedule (SCED) |

**Tabela 2- Direcionadores de custo. Adaptado de [18].**

Como na versão inicial, o fator de ajuste do esforço (EAF) é o produto dos direcionadores de custo. Cada direcionador de custo tem fatores de ponderação individual para cada nível de classificação. Os fatores de ponderação foram determinados por entrevistas com *experts* e acordados em reuniões de pesquisa. A tabela 3 mostra um exemplo dos valores para o direcionador de custo Capacidade de Análise (*Analyst Capability*). Existe uma tabela assim para cada direcionador de custo. O estimador tem que selecionar o respectivo nível para cada direcionador de custo no projeto.

| **ACAP Descriptors:** | $15^{th}$ percentile | $35^{th}$ percentile | $55^{th}$ percentile | $75^{th}$ percentile | $90^{th}$ percentile | |
|---|---|---|---|---|---|---|
| **Rating Levels** | Very Low | Low | Nominal | High | Very High | Extra High |
| **Effort Multipliers** | 1.42 | 1.19 | 1.00 | 0.85 | 0.71 | n/a |

**Tabela 3 - Capacidade de Análise [18].**

Em projetos pequenos a deseconomia de escala é balanceada, assim o fator $E$ pode ser negligenciadao ($E = 1$). Porém em projetos médios e de grande porte onde a dessecotomia de escada exerce grande influência, o fator $E$ é determinado usando a seguinte fórmula:

**Equação 5**

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j$$

Onde $B = 0.91$ para o COCOMO II.

A constante $B$ pode ser calibrada ao ambiente de desenvolvimento. A influência dos fatores de escala é uma das maiores diferenças entre o COCOMO 81 e o COCOMO II. No COCOMO 81 os fatores de escala são aplicados no código do projeto inteiro, já no COCOMO II os fatores de escala são só aplicados ao tamanho do código novo e modificado. Para o COCOMO 81, se aplicar os fatores de escala a muitas milhas de LOC do produto produz uma estimativa muito grande, porém a maior parte do produto não está sendo afetada pela mudança [18].

Tabela 8 contem os cinco fatores de escala (SF) que determinam o $E$:

| | |
|---|---|
| - Precedentedness (PREC) | - Team Cohesion (TEAM) |
| - Development Flexibility (FLEX) | - Process Maturity (PMAT) |
| - Architecture / Risk Resolution (RESL) | |

**Tabela 4 - Fatores de Escala [18].**

Cada fator de escala tem uma faixa de valores que variam de muito pouco e extra alto. O fator extra alto é sempre 0.00 o que faz a constante $E = 0.91$. A premissa é que um extra alto leva a uma "economia" de escala para o projeto.

A versão inicial do COCOMO II prove uma simples equação para estimar o cronograma como segue:

**Equação 6**

$$TDEV = [\, C * (PM_{ns})^{D+0.2*(E-B)}] * \frac{SCED\%}{100}$$

Onde $C = 3.67, D = 0.28, B = 0.91$ para o COCOMO II.

$C$ é um coeficente de cronograma e pode ser calibrado, $B$ é o mesmo do esforço, $D$ é uma constante do cronograma e pode ser calibrada, $E$ é o mesmo da equação do esforço. $PM_{ns}$ é pessoas mês estimada sem o direcionador de custo SCED, $SCED\%$ é a porcentagem de compressão do cronograma relativo ao cronograma nominal, $TDEV$ é o tempo de desenvolvimento.

### 4.1.1 Uma Alternativa a Entrada de Tamanho para o COCOMO II: Pontos de Função

Como o COCOMO usa KDSI (*Kilo Delivered Source Instruction)* como entrada de tamanho. Uma desvantagem do KDSI é a dificuldade de estimar o número de LOC no começo do projeto. Para resolver esse problema, pontos de função podem ser usados como uma alternativa.

Diferentes linguagens de programação necessitam de diferentes quantidades de LOC para realizar uma funcionalidade específica, existem tabelas de conversão de pontos de função para LOC[3]. Os valores de PF/LOC são determinados por análise estatística de um conjunto de diferentes projetos.

## 4.2 MODELO DE PUTNAM – SLIM

O modelo de Putnam [37] é um modelo formal de estimativa que foi desenvolvido na década de 70 por Lawrence Putnam. O modelo é usado para estimar o esforço e a duração do desenvolvimento de software. Como a maioria dos modelos paramétricos ele também usa fórmulas matemáticas para estimar o esforço integrando o tamanho e outras características do projeto para realizar a estimativa. O modelo é um modelo de macro estimação e foi calibrado usando regressão sobre dados históricos de projetos.

Enquanto trabalhava como engenheiro na *General Eletric* e nas forças armadas americanas, Putnam descobriu que o nível da equipe em um software segue uma distribuição de *Rayleigh*. Putnam descreve esta observação usando a seguinte equação:

[3] *No endereço http://www.qsm.com/resources/function-point-languages-table pode ser encontrada uma lista atualizada.*

**Equação 7**

$$\frac{B^{1/3} * Tam}{Produtividade} = Esforço^{1/3} * Tempo^{4/3}$$

Que pode ser transformada em:

**Equação 8**

$$Esforço = \left[\frac{Tam}{Produtividade * Tempo^{\frac{4}{3}}}\right]^3 * B$$

Onde o esforço total é em pessoas ano (diferente do COCOMO que é em pessoas mês). Para o fator Tamanho qualquer unidade de medida pode ser usada desde que seja a mesma unidade incluída na Produtividade. (Putnam usa SLOC Efetivas). Produtividade é a habilidade das organizações de software desenvolver um sistema a uma taxa fixa de defeitos. Tempo é o cronograma total do projeto em anos. $B$ é um fator de escala que é dependente do tamanho do projeto.

O modelo de Putnam necessita de menos parâmetros de entrada que o COCOMO, mas é projetado para ser usado somente em grandes projetos que excedam as 70.000 SLOC [38]. A grande vantagem desse modelo é ser relativamente fácil para as organizações calibrarem com seu ambiente desde que o Tamanho, Esforço e duração (Tempo) sejam medidas comuns e de fácil manutenção na organização. Como é fácil determinar a média da Produtividade através do tempo esse fato possibilita estimar o esforço para projetos futuros. A fórmula para encontrar a produtividade é derivada de uma simples transformação algébrica da fórmula anterior:

**Equação 9**

$$Produtividade = \frac{Tam}{\left[\frac{Esforço}{B}\right]^{1/3} * Tempo^{4/3}}$$

SLiM é o nome de uma ferramenta comercial que implementa o modelo de Putnam. Essa ferramenta é desenvolvida e distribuída pela QSM Inc, uma companhia fundada por Lawrence Putnam. SLiM foi um dos primeiros modelos paramétricos de esforço implementados, mas não é mais popular atualmente. Um dos motivos pode ser que o SLIM

não é público como o COCOMO, ele é comercializado. Outra razão é que o interesse em pesquisar o modelo caiu muito desde a década de 90 [38].

## 4.3 SEER-SEM

SEER-SEM é um modelo paramétrico comercial desenvolvido pela *Galorath Inc.* O desenvolvimento do SEER-SEM começou no final da década de 1980 e hoje muitos governos e projetos comerciais usam SEER-SEM para estimar seus custos. Como o modelo é desenvolvido por uma empresa privada não há muitas informações sobre as relações usadas para estimar os custos. Fischman et al. [39] dá uma introspecção conceitual e técnica do modelo, porém mesmo assim muitos dados ainda são omitidos.

## 4.4 *PRICE SISTEMS - TRUEPLANNING*

*TruePlanning* é um software comercial de estimativa de custos desenvolvido pela *Price Systems*. O *TruePlanning* é um conjunto de ferramentas que não contém exclusivamente um modelo paramétrico, mas também encorpa fermentas que suportam julgamento de especialistas usando *bottom-up* e *top-down* como metodologia. O modelo não é de acesso publico. Por isso as fórmulas não são divulgadas. O *TruePlanning* é uma das ferramentas comercias de estimativa de custo mais usadas pelo governo dos Estados Unidos [38].

# 5 TÉCNICAS PARA ESTIMAR O TAMANHO, ESFORÇO E CRONOGRAMA.

Nesse capítulo serão apresentadas algumas das técnicas mais usadas e difundidas para se estimar o tamanho, esforço e cronograma no desenvolvimento de software. A seção também abordará possíveis cenários para utilizá-las.

## 5.1 TAMANHO

O propósito de uma estimativa de tamanho é apoiar a previsibilidade de longo alcance na parte larga do Cone da Incerteza.

As medidas de tamanho comum de LOC e pontos de função têm diferentes pontos fortes e fracos, assim como medidas personalizadas definidas por organizações para uso próprio. Criação de estimativas usando medidas de tamanho múltiplo e em seguida, olhando para a convergência tende a produzir resultados mais precisos [2].

Há inúmeras medidas de tamanho como: *User stories*, *Story points*, Requisito, Casos de Uso, Pontos de Função, Paginas Web, Componentes de interface gráfica, Tabelas do Banco de Dados, Definição de interface, Classes, Funções/sub-rotinas, LOC e etc.

Dentre elas as mais usadas são LOC e Pontos de Função [43] e essas serão tratadas com mais detalhe, também será abordada a técnica de Pontos Por Casos de Uso que vem ganhando espaço como métrica de tamanho [40].

### 5.1.1 Papel de Linhas de Código na Estimativa de Tamanho

O uso de LOC na estimativa de tamanho é uma faca de dois gumes. Do lado positivo podem-se enumerar as seguintes vantagens [2]:

- É fácil a coleta em LOC de projetos passados.
- Já existem muitos dados sobre LOC em muitas organizações.
- Esforço em LOC são aproximadamente constantes em linguagens de programação (esforço em linhas de código é mais em função do

tamanho do projeto e tipo de software do que da linguagem de programação).

- Muitas ferramentas comerciais baseiam suas estimativas em linhas de código.

Do lado negativo as LOC apresentam muitas dificuldades como métrica de tamanho:

- Medidas simples como “linhas de código por pessoa” esta sujeita a erros devido à deseconomia de escala do software e da taxa de codificação de cada tipo de software.
- LOC não podem ser usadas como base para estimar atribuições de tarefas, pois há diferenças na produtividade entre os programadores.
- Não é intuitivo usar LOC para estimar os requisitos, design, e outras atividades que precedem a criação do código.
- Dependente da Tecnologia.

### 5.1.2 Tamanho Funcional

Uma alternativa para a medida LOC é pontos de função. Um ponto de função é uma estimativa aproximada de uma funcionalidade entregue em um projeto de software sob a visão do usuário, ele pode ser usado para estimar o tamanho em estágios iniciais de um projeto [41]. Pontos de função são mais fáceis de calcular a partir de uma especificação de requisitos do que linhas de código são, e eles fornecem uma base para computar o tamanho em linhas de código. Existem muitos métodos diferentes para a contagem de pontos de função conhecidos como FSM (*Functional size measurement*), métricas de tamanho funcional. O padrão mais usado para a contagem de função de ponto de é mantido pelo *Internatinal Function Point International Users Group* (IFPUG).

Nas últimas três décadas o uso de métricas de tamanho funcional tem mostrado que é atualmente o único método comprovado de dimensionamento de software que dá resultados consistentes e confiáveis para estimativa do projeto e comparações de produtividade [40]. O método FSM para o dimensionamento é apoiado e continuamente aprimorado pela comunidade internacional.

Atualmente são cinco métodos FSM reconhecidos pela *Internatinal Organization for Standartization* (ISO).

- COSMIC-FFP 3.0 - ISO/IEC 19761:2010.
- FiSMA FSM 1.1 ISO/IEC 29881:2008
- IFPUG CPM 4.3 ISO/IEC 20926:2009
- MK II 1.3.1 Não-ajustado ISO/IEC 20968:2002
- NESMA FPA 2.1 Não-Ajustado ISO/IEC 24570:2005

Os métodos da IFPUG e NESMA são os mais difundidos [3], e serão abordados nos próximos tópicos.

### 5.1.3 Análise de Pontos de Função – IFPUG

A análise de ponto de função (APF) é uma técnica para medir o tamanho de aplicações. Ela foi criada para suprir as dificuldades associadas com a medição por linhas de código (LOC) e para desenvolver um mecanismo para prever o esforço associado ao desenvolvimento de software.

Uma vez que ponto de função mede um sistema a partir de uma perspectiva funcional, esta técnica é independente de tecnologia. Ou seja, o número de pontos de função de um sistema será sempre o mesmo, não importando a linguagem, metodologia de desenvolvimento, ambiente ou hardware utilizado. A única variável é o esforço necessário para entregar um conjunto de pontos de função. Portanto, a análise de pontos de função pode ser utilizada para determinar quando uma ferramenta, ambiente, linguagem, etc., é mais produtiva que outra dentro ou entre organizações. Este é um ponto crítico e um dos maiores valores da análise de pontos de função [3].

Segundo Longstreet [42], os componentes na APF são divididos em dois grupos: Funções de Dados, que representam a funcionalidade fornecida ao usuário para encontrar dados internos e externos. Funções de Transição, que representam a funcionalidade fornecida ao usuário para processar dados. Essas categorias por sua vez são classificadas da seguinte forma [43]:

**Função de Dados**: classificadas como Arquivos Lógicos Internos (ALI) e Arquivos de Interface Externa (AIE).

**Função de Transação**: classificadas em Entradas Externas (EE), Saídas Externas (SE) e Consultas Externas (CE).

Nesta etapa as funcionalidades da aplicação começam a ser identificadas e contadas.

As Funções Tipo Dados representam as funcionalidades fornecidas pelo sistema ao usuário, para atender às necessidades referentes aos dados que o sistema irá manipular.

Essas funções podem ser:

**Arquivo Lógico Interno**: grupo logicamente relacionado de dados ou informações de controle, identificável pelo usuário, mantido dentro da fronteira da aplicação que está sendo controlada. Por exemplo, as tabelas ou classes do sistema.

**Arquivo de Interface Externa**: grupo logicamente relacionado de dados ou informações de controle, referenciado pela aplicação, identificável pelo usuário, mantido fora da fronteira da aplicação que está sendo controlada. Por exemplo, as tabelas acessadas em outro sistema.

A fronteira da aplicação indica a separação entre o projeto que está sendo medido e as aplicações externas ao domínio do usuário. É através dela que torna-se possível definir quais funcionalidades serão incluídas no processo de contagem dos pontos de função.

Cada Arquivo Lógico Interno e cada Arquivo de Interface Externa possuem dois tipos de elementos que devem ser contados para cada função identificada:

**Tipos de Elementos de Dados**: campo único, reconhecido pelo usuário, não recursivo. Por exemplo: campos das tabelas.

**Tipos de Elementos de Registros**: subgrupo de dados, reconhecido pelo usuário. Por exemplo: generalização/especialização de classes.

As Funções Tipo Transação representam as funcionalidades de processamento dos dados fornecidas pelo sistema ao usuário. Essas funções podem ser:

**Entrada Externa**: processo elementar da aplicação que processa dados ou informações de controle que vêm de fora da fronteira da aplicação que está sendo controlada.

Exemplos: validações, fórmulas e cálculos matemáticos cujos parâmetros vêm de fora da fronteira da aplicação.

**Saída Externa**: processo elementar da aplicação que geram dados ou informações de controle que são enviados para fora da fronteira da aplicação que está sendo controlada. Exemplos: relatórios e gráficos.

**Consulta Externa:** processo elementar da aplicação que representa uma combinação de entrada (solicitação de informação) e saída (recuperação de informação). Exemplos: consultas implícitas, verificação de senhas e recuperação de dados com base em parâmetros.

Cada EE, SE e CE possui dois tipos de elementos que devem ser contados para cada função identificada:

**Tipos de Elementos de Dados**: campo único, reconhecido pelo usuário, não recursivo. Por exemplo: campos das tabelas.

**Tipos de Arquivos Referenciados** ou **Arquivos Referenciados**: arquivos lógicos utilizados para processar a entrada e/ou saída. É o total de ALI e AIE utilizados pela transação.

Ao final dessa etapa devem estar identificadas quantas EE, SE, e CE o sistema possui e, para elas, quantos são os Tipos de Elementos de Dados e os Arquivos Referenciados encontrados.

Todos os componentes citados anteriormente são classificados como, *High* (complexo), *Low* (simples) e *Avarage* (médio). Os componentes EE, SE, CE são classificados segundo o número de arquivos atualizados ou referenciados e o número de Tipos de Elementos de Dados. Os componentes AIE e ALI são classificados segundo ao número de Tipos de Elementos de Dados e a quantidade de Registro do Tipo de Elemento (RTE). RTE é um subgrupo de elementos de dados dentro de um ALI ou AIE. Tipo de Elemento de Dados é um campo não recursivo dentro de um ALI ou AIE.

A figura 15 mostra a classificação dos componentes:

| Entradas Externas | | | |
|---|---|---|---|
| ALI e AIE | Elementos de Dados | | |
| | 1 – 4 | 5 – 15 | > 15 |
| 0 - 1 | Simples | Simples | Médio |
| 2 – 3 | Simples | Médio | Complexo |
| > 3 | Médio | Complexo | Complexo |

| Saídas Externas e Consultas Externas | | | |
|---|---|---|---|
| ALI e AIE | Elementos de Dados | | |
| | 1 – 5 | 6 – 19 | > 19 |
| 0 - 1 | Simples | Simples | Médio |
| 2– 3 | Simples | Médio | Complexo |
| > 3 | Médio | Complexo | Complexo |

| ALI e AIE | | | |
|---|---|---|---|
| Registros | Tipos de Elementos | | |
| | 1 – 19 | 20 -50 | > 50 |
| 1 | Simples | Simples | Médio |
| 2 – 5 | Simples | Médio | Complexo |
| > 5 | Médio | Complexo | Complexo |

| Valores para transações | | | |
|---|---|---|---|
| Classificação | Valores | | |
| | SE | CE | EE |
| Baixo | 4 | 3 | 3 |
| Médio | 5 | 4 | 4 |
| Complexo | 7 | 6 | 6 |

| Valores para ALI e AIE | | |
|---|---|---|
| Classificação | Valores | |
| | ALI | AIE |
| Simples | 7 | 5 |
| Médio | 10 | 7 |
| Complexo | 15 | 10 |

**Figura 15 - Classificação dos componentes da APF**

E por fim chega-se a seguinte tabela:

| Pontos de Função | | | | Contagem |
|---|---|---|---|---|
| **Características do Programa** | **Simples** | **Médio** | **Complexo** | |
| EE | __ × 3 | __ × 4 | __ × 6 | |
| SE | __ × 4 | __ × 5 | __ × 7 | |
| CE | __ × 3 | __× 4 | __ × 6 | |
| ALI | __ × 7 | __ × 10 | __ × 15 | |
| AIE | __ × 5 | __ × 7 | __ × 10 | |
| TOTAL | | | | |

**Tabela 5 - Cálculo dos PF não ajustados**

### Fator de ajuste

Ainda existe um fator de ajuste para os pontos de função contados em um sistema ou aplicação. O Fator de Ajuste é baseado em 14 características gerais de sistemas que dizem respeito a funcionalidades gerais da aplicação que está sendo analisada [40] [42]

[43]. Enquanto as funções do tipo dado refletem requisitos específicos de armazenamento e as funções do tipo transação requisitos específicos de processamento, as características gerais refletem funções que afetam a aplicação de maneira geral [43]. Cada característica possui uma descrição associada que ajuda a determinar o grau de influência de cada característica [40]. O grau de influência é classificado em uma escala de 0 a 5, que são: 0. Não está presente, ou não exerce influência; 1. Influência casual; 2. Influência Moderada; 3. Influência Média; 4. Influência significativa; 5. Forte influência, em toda parte.

A tabela 6 apresenta as 14 características gerais de sistemas que são utilizadas para determinar o fator de ajuste.

| Característica do Sistema. | | Breve Descrição |
|---|---|---|
| 1 | Comunicação de dados. | Quantos canais de comunicação que existem para auxiliar na transferência ou troca de informações com a aplicação ou sistema? |
| 2 | Distribuição de processamento. | Como são distribuídos os dados e o processamento das funções de manipulação. |
| 3 | Performance. | O usuário necessita de um tempo de resposta? |
| 4 | Exploração da configuração. | Como a configuração e hardware atual de onde a aplicação é executada são explorados? |
| 5 | Taxa de transações. | Com que freqüência, transações são executadas, diariamente, semanalmente, mensalmente, etc.? |
| 6 | Entrada de dados on-line. | Qual a porcentagem de informação é de origem on-line? |
| 7 | Eficiência do usuário final. | A aplicação foi desenvolvida para usuários com conhecimentos avançados? |
| 8 | Atualização on-line. | Quantos ALI são atualizados através de transações on-line? |
| 9 | Processamento Complexo. | A aplicação possui processamento lógico ou matemático extensivo? |
| 10 | Reusabilidade. | A aplicação foi desenvolvida para atender uma ou várias necessidades do usuário? |
| 11 | Facilidade de instalação. | Quão fácil é a conversão ou instalação da aplicação. |
| 12 | Facilidade Operacional. | Quão efetivo e/ou automático são os procedimentos de start-up, backup e recuperação? |
| 13 | Várias localidades. | A aplicação foi desenvolvida e oferece suporte para ser instalada em vários locais de várias organizações? |
| 14 | Facilidade de mudança. | A aplicação foi especificamente projetada, desenvolvida e oferece suporte para facilitar mudanças? |

**Tabela 6 - Fatores de ajuste dos PF.**

Uma vez que as 14 questões forem respondidas com as respostas classificadas de 0 a 5, o fator de ajuste é calculado utilizando a equação Valor do Fator de Ajuste (VAF – *Value Adjustment Factor*) que consiste em [43]:

**Equação 10**

$$VAF = (TDI * 0.01) + 0.65$$

Onde TDI é a soma dos resultados de todas as 14 características.

O cálculo do fator de ajuste pode influenciar no tamanho em ± 35% [43].

Após o cálculo do VAF os PF ajustados são calculados segundo a fórmula:

**Equação 11**

$$PF_{ajustados} = PF_{nãoAjustados} * VAF$$

### 5.1.4 NESMA - *The Netherlands Software Metrics Association*

A contagem de Ponto de função requer atravessar a especificação de requisitos linha por linha e, literalmente, contando cada entrada, saída, arquivos, e assim por diante. Isso pode ser demorado. Sem contar que no início do projeto a especificação pode estar incompleta.

A NESMA reconhece três tipos de contagem de pontos de função [44]: contagem de pontos de função detalhada, contagem de pontos de função estimativa e a contagem de pontos de função indicativa.

Os métodos estimativo e indicativo para a contagem de pontos de função foram desenvolvidos pela NESMA para permitir que uma contagem de pontos de função seja feita nos momentos iniciais do ciclo de vida de um sistema. A contagem indicativa da NESMA é também conhecida como "método holandês".

**A contagem detalhada de pontos de função**

A contagem detalhada é a contagem usual de pontos de função e é realizada da seguinte forma: Determinam-se todas as funções de todos os tipos (ALI, AIE, EE, SE, CE), determina-se a complexidade de cada função (Baixa, Média, Alta) e calcula-se o total de pontos de função não ajustados.

**A contagem estimativa de pontos de função**

A contagem estimativa é realizada da seguinte forma: Determina-se todas as funções de todos os tipos (ALI, AIE, EE, SE, CE), toda função do tipo dado (ALI, AIE) tem sua complexidade funcional avaliada como Baixa, e toda função transacional (EE, SE, CE) é avaliada como de complexidade média e então calcula-se o total de pontos de função não ajustados.

Logo, a única diferença em relação à contagem usual de pontos de função é que a complexidade funcional não é determinada individualmente para cada função, mas pré-definida para todas elas.

**A contagem indicativa de pontos de função**

A contagem indicativa é realizada da seguinte forma: determina-se a quantidade das funções do tipo dado (ALIs e AIEs), calcula-se o total de pontos de função não ajustados da aplicação da seguinte forma:

**Equação 12**

$$Tamanho\ indicativo\ PF\ =\ 35 * n\acute{u}mero\ de\ ALIs\ +\ 15 * n\acute{u}mero\ de\ AIEs$$

Portanto esta estimativa é baseada somente na quantidade de arquivos lógicos existentes (ALIs e AIEs).

A contagem indicativa é baseada na premissa de que existem aproximadamente três EEs (para adicionar, alterar, e excluir dados do ALI), duas SEs, e uma CE na média para cada ALI, e aproximadamente uma SE e uma CE para cada AIE.

Os números 35 e 15 foram obtidos por meio de calibração, eles podem ser alterados conforme o ambiente em que são usados.

Em uma seleção de 127 projetos do banco de dados do ISBSG [3], foi encontrado o coeficiente de correlação (R de Pearson) de 78,4% entre o tamanho real em PF Não ajustados e o estimado por uma aproximação da Contagem Indicativa NESMA.

### 5.1.5 Pontos por Casos de Uso

É uma variação da contagem de Pontos de Função específica para medição do tamanho de Projetos Orientados a Objetos. Explora o modelo e a descrição através das funcionalidades contidas nos Casos de Uso [45].

No início da década de 90 começou um movimento de que APF não servia para medir projetos orientados a objetos. Dentro desse contexto, em 1993 Gustav Karner da *Objective Systems SF AB* propôs em um trabalho sobre a metodologia dos Pontos por Casos de Uso baseado em APF [43], os passos foram relacionados, com o intuito de estimar recursos para projetos orientados a objetos já na fase de levantamento de Casos de Uso.

O processo de medição de Pontos por Caso de Uso (UCP – *Use Case Points*) consiste resumidamente em 6 passos [45]:

1. Contar os atores e identificar sua complexidade.
2. Contar os Casos de Uso e identificar sua complexidade.
3. Somar o total de atores e o total de Casos de Uso para obter os UCP não ajustados.
4. Determinar a complexidade do fator técnico.
5. Determinar a complexidade do fator ambiental.
6. Calcular UCP ajustado.

**Cálculo dos pesos dos atores**

O UAW (*Unadjusted Actor Weight*) é o Peso total dos atores no sistema. Para calcular o UAW, os atores devem ser classificados como simples, médio ou complexo com base em suas interações.

O UAW é calculado pela contagem do número de atores em cada categoria, multiplicando cada pelo seu fator de ponderação especificado e, em seguida, somando os produtos conforme a tabela 7 [3] [45].

| Tipo de Ator | Peso | Descrição |
|---|---|---|
| Ator Simples | 1 | Outro sistema acessado através de uma API de programação |
| Ator Médio | 2 | Outro sistema interagindo através de um protocolo de comunicação, como TCP/IP ou FTP |
| Ator Complexo | 3 | Um usuário interagindo através de uma interface gráfica (stand-alone ou Web) |

**Tabela 7 - Pesos de Atores**

**Cálculo dos pesos dos casos de uso**

O UUCW (*Unadjusted Use Case Weight*) é o cálculo inicial do peso bruto dos casos de uso. Ele é calculado pela contagem do número de casos de uso em cada categoria (simples, médio, complexo) de acordo com a tabela 8, então multiplica-se cada categoria de caso de uso com o seu peso, e, em seguida, somam-se os resultados [3][45].

| Tipo de Caso de Uso | Número de Transações/Entidades | Peso |
|---|---|---|
| Simples | Até 3 de transação ou até 5 classes de análise | 5 |
| Médio | 4 a 7 transações ou de 5 a 10 classes de análise | 10 |
| Complexo | 7 ou mais transações ou mais de 10 classes de análise | 15 |

**Tabela 8 - Pesos das transações/entidades**

O UUCP (*Unadjusted Use Case Points*) é o peso total não ajustado. É calculado pelo somatório entre os Pesos dos atores e Casos de uso:

**Equação 13**

$$UUCP = UAW + UUCW.$$

**Cálculo do fator de ajuste**

O método de ajuste é bastante similar ao adotado pela técnica de Pontos de Função, e é constituído de duas partes [3] [45]:

Cálculo de fatores técnicos, cobrindo uma série de requisitos funcionais do sistema;

Cálculo de fatores de ambiente, requisitos não-funcionais associados ao processo de desenvolvimento tais como experiência da equipe, motivação e estabilidade do

projeto. Estes dois fatores geram multiplicadores distintos, que devem ser aplicados ao peso total não-ajustado (UUCP), calculado anteriormente.

**Fatores técnicos**

Para calcular o fator de complexidade técnica do sistema (TCF - *Technical Complexity Factor*), utiliza-se a tabela 9.

| Fator | Requisito | Peso |
|---|---|---|
| T1 | Sistema distribuído | 2 |
| T2 | Tempo de Resposta | 2 |
| T3 | Eficiência | 1 |
| T4 | Processamento complexo | 1 |
| T5 | Código reusável | 1 |
| T6 | Facilidade de instalação | 0.5 |
| T7 | Facilidade de uso | 0.5 |
| T8 | Portabilidade | 2 |
| T9 | Facilidade de mudança | 1 |
| T10 | Concorrência | 1 |
| T11 | Recursos de segurança | 1 |
| T12 | Acessível por terceiros | 1 |
| T13 | Requer treinamento especial | 1 |

**Tabela 9 - Pesos dos fatores Técnicos**

Para cada fator listado na tabela Fator e Requisito, será atribuído um valor que determina a influência do requisito no sistema, variando entre 0 a 5, sendo que o valor 0 indica nenhuma influência, 3 indica influência moderada e 5 indica forte influência.

O cálculo do TCF é realizado da seguinte forma [45]:

Cada um dos fatores é calculado assim:

**Equação 14**

$$Tn = Pn * In$$

Onde $n$ é o número de cada um dos fatores, $P$ é o peso (tabela) e $I$ é a influência.

$TFactor$ é o somatório de T1 a T13. O cálculo final do TCF:

**Equação 15**

$$TCF = 0.6 + (0.01 * TFactor).$$

**Fatores ambientais**

O ECF (*Environmet Complexity Factor)* verifica as características da equipe e do ambiente em que será desenvolvido o projeto. Os requisitos e os pesos são obtidos da tabela 10.

| Fator | Requisito | Peso |
|---|---|---|
| E1 | Familiaridade com RUP ou outro processo formal | 1.5 |
| E2 | Experiência com a Aplicação em desenvolvimento | 0.5 |
| E3 | Experiência em Orientação a Objetos | 1 |
| E4 | Presença de analista experiente | 0.5 |
| E5 | Motivação | 1 |
| E6 | Requisitos estáveis | 2 |
| E7 | Desenvolvedores em meio-expediente | -1 |
| E8 | Linguagem de programação difícil | 2 |

**Tabela 10 - Pesos dos fatores ambientais**

No caso dos Fatores Ambientais, o nível de influência indica o nível de disponibilidade de cada recurso no decorrer do projeto: desta forma, determinar que um dado fator tem nível de influência alto (isto é, atribuir a ele o valor 5) significa dizer que este fator está presente no projeto como um todo e influencia seu desenvolvimento. Da mesma forma, atribuir um valor de influência zero (nenhuma influência) a um fator indica que o mesmo não está presente no processo de desenvolvimento.

O cálculo do ECF é realizado da seguinte forma [45]:

Cada um dos fatores é calculado assim:

**Equação 16**

$$En = Pn * In$$

Onde $n$ é o número de cada um dos fatores, $P$ é o peso (tabela) e $I$ é a influência.

$EFactor$ é o somatório de E1 a E8. O cálculo final do ECF:

**Equação 17**

$$ECF = 0.6 + (0.01 * EFactor).$$

Finalmente, UCP (ajustado) são calculados usando a seguinte fórmula:

**Equação 18**

$$UCP = UUCP * TCF * ECF$$

**Estimativa de cronograma usando UCP**

Alguns autores sugerem a utilização de 20 pessoas/hora por unidade de UCP. Outros sugerem o seguinte refinamento [45].

X = total de itens E1 a E6 com pontuação menor que 3.

Y = total de itens E7 a E8 com pontuação menor que 3.

Se $X + Y \leq 2$ usar 20 como unidade de homens/hora (o valor deve estar entre 15 e 30, e pode ser ajustado seguindo dados históricos).

Se $3 \leq X + Y \leq 4$, usar 28 como unidade homem/hora.

Se $X + Y \geq 5$, deve-se tentar modificar o projeto de forma a baixar o número, pois o risco de insucesso é grande.

Então a estimativa de cronograma é calculada pela fórmula:

**Equação 19**

$$UCP * pessoa/hora\ por\ unidade\ de\ UCP.$$

## 5.2 ESFORÇO

Muitos projetos eventualmente estimam o esforço através de uma lista de tarefas detalhadas. Mas no inicio do projeto, a estimativa de esforço é mais precisa quando computada da estimativa de tamanho.

A principal influência no esforço em um projeto é o tamanho do software que será construído. Em segundo lugar vem a produtividade da organização [2].

Caso não haja dados históricos da produtividade da organização pode ser usada a média de produtividade da indústria para aproximar a produtividade, na indústria contém valores médios de diferentes tipos de software como: sistemas internos de negocio, sistemas críticos, jogos, drivers de dispositivos, e assim por diante. Devido a isso deve-se ter cuidado ao utilizar tais dados pois há um fator de diferença 10 na produtividade de organizações diferentes dentro da indústria [2].

### 5.2.1 Computando o Esforço a Partir do Tamanho usando Comparação Informal dos Projetos Passados.

Se os dados históricos são de projetos dentro de uma faixa estreita de tamanho (um fator de 3 de diferença do menor para o maior), provavelmente haverá sucesso usando um modelo linear para calcular a estimativa de esforço para um novo projeto com base nos resultados do esforço de projetos semelhantes anteriores [2]. Caso os dados históricos fujam disso é necessário usar um modelo mais formal para estimar o esforço para obter melhores resultados. [2]

Em projetos maiores que 1.000 pontos de função ou 100.000 linhas de código estimar usando comparação informal ou métodos manuais induz um erro significativo na estimativa, e a não utilização de uma software sofisticado que implementa modelos formais para estimar projetos maiores que 5.000 pontos de função ou 500.000 linhas de código é considerada uma má pratica de gestão [2].

### 5.2.2 Métodos do ISBSG

O *International Software Benchmarking Standards Group* (ISBSG) desenvolveu um método interessante e útil para computar o esforço baseado em três fatores: o tamanho de um projeto em pontos de função, o tipo de ambiente de desenvolvimento e o tamanho máximo da equipe [40]. Apresentado por tipo de projeto, as seguintes oito equações são as mais usadas para estimar esforço usando essa abordagem. As equações produzem uma estimativa em pessoa mês, assumindo 132 projetos com foco horas por pessoa mês (ou seja, excluindo férias, feriados, dias de treinamento, reuniões de empresas, e assim por diante). A fórmula geral é uma fórmula para uso em todos os tipos de projeto e é baseada em dados de calibração a partir de cerca de 600 projetos. As outras categorias são calibradas com dados de 63-363 projetos [40].

Tipo do Projeto:

Geral:

**Equação 20**

$$PM = 0.512 * PF^{0.392} * TamMaxEquipe^{0.791}$$

*Mainframe*:

**Equação 21**

$$PM = 0.685 * PF^{0.507} * TamMaxEquipe^{0.464}$$

Médio:

**Equação 22**

$$PM = 0.472 * PF^{0.375} * TamMaxEquipe^{0.882}$$

*Desktop*:

**Equação 23**

$$PM = 0.157 * PF^{0.591} * TamMaxEquipe^{0.810}$$

Linguagens de terceira geração:

**Equação 24**

$$PM = 0.425 * PF^{0.488} * TamMaxEquipe^{0.697}$$

Linguagens da quarta geração:

**Equação 25**

$$PM = 0.317 * PF^{0.472} * TamMaxEquipe^{0.784}$$

Melhoramento:

**Equação 26**

$$PM = 0.669 * PF^{0.338} * TamMaxEquipe^{0.758}$$

Novo desenvolvimento:

**Equação 27**

$$PM = 0.520 * PF^{0.385} * TamMaxEquipe^{0.866}$$

Um aspecto interessante do método ISBSG é que as fórmulas para o esforço dependem do tamanho máximo da equipe do projeto, com equipes menores produzindo menores estimativas de esforço total. Do ponto de vista de controle do projeto, essa diferença pode levar a usar um tamanho menor de equipe em vez de um maior. O ideal é usar o método de estimativa do ISBSG para fazer uma estimativa mais geral do esforço e combinar com estimativas de outros métodos e observar a convergência ou dispersão entre elas e dar maior peso as técnicas com maior precisão [2].

### 5.2.3 Relações Lineares

Para se estimar o esforço de forma menos empírica, deve usar a definição de Produtividade (P): razão de bens ou serviços produzidos (F) por unidade de trabalho e custo (E).

**Equação 28**

$$P = \frac{F}{E}$$

O conceito de produtividade depende principalmente da determinação do tamanho a ser utilizado [43]. Por exemplo, usando PF, pode-se definir a produtividade (P) como, razão entre o tamanho do projeto em PF e a quantidade total de horas (H) ou de homens * mês (HM) gastos no desenvolvimento, na unidade $PF/H$ ou $PF/HM$ respectivamente. A taxa de entrega, geralmente é a medida inversa $H/PF$ e $HM/PF$ respectivamente. Ambos os indicadores representam a mesma relação entre o tamanho e o esforço. Então para fins de esforço usa-se a equação 28 onde F é uma medida em PF.

Contudo, para efeito de estimativa de esforço, o que parece ser mais natural é a utilização da taxa de entrega (T).

**Equação 29**

$$E = F * T$$

Em que é possível obter uma relação direta entre o aumento da taxa de entrega e o aumento do esforço.

## 5.3 CRONOGRAMA

A necessidade de cumprir prazos dos clientes, prazos de feiras, prazos regulamentares, e outros tipos de prazos parecem colocar grande parte da pressão na estimativa de cronograma. A estimativa do cronograma parece produzir maior parte das discussões sobre estimação.

Uma vez que se passa de abordagens intuitivas para as abordagens baseadas em dados históricos, a estimativa de cronograma torna-se um cálculo simples, que flui da estimativa de tamanho e esforço.

### 5.3.1 Equação Básica do Cronograma

A regra é que se pode estimar o cronograma no início de um projeto usando uma equação básica de cronograma [2] [25]:

**Equação 30**

$$Cronograma\ em\ Meses = 3.0 * PessoasMês^{\frac{1}{3}}$$

Às vezes, o fator 3.0 pode ser 2.0, 2.5, 4.0 ou um número semelhante (pode ser obtido por meio da calibração com dados históricos), mas a ideia básica de que o cronograma é uma função raiz cubica do esforço é quase universalmente aceita pelos peritos em estimativa. Boehm em 1981 comentou que esta fórmula foi um dos resultados mais replicados em engenharia de software. E os estudos mais recentes continuam validando essa equação [6] [18].

MacConnell [2] recomenta usar essa equação para estimar o cronograma no início de projetos de médio a grande porte.

A equação de cronograma implicitamente assume que o tamanho da equipe possa ser ajustado ao tamanho implícito pela equação. Se o tamanho da equipe for fixo, o cronograma não vai variar em proporção à raiz cúbica do esforço, irá variar de forma mais ampla com base nas restrições de equipe.

A equação de cronograma básica não se destina para a estimativa de pequenos projetos ou fases finais de grandes projetos. [2]

### 5.3.2 Calculando Cronograma Usando Comparação Informal com Projetos Passados

William Roetzheim propôs estimar o cronograma de novos projetos com base em uma relação do cronograma e esforço de projetos anteriores [6].

Roetzheim sugere estimar cronograma em meses usando a fórmula para Comparação Informal Projetos Passados:

**Equação 31**

$$Cronograma = Cronograma_{projetosPassados} * \left(\frac{Esforço_{Estimado}}{Esforço_{projetosPassados}}\right)^{1/3}$$

O expoente de $1/3$ é usado para projetos de médio à grande porte (cerca de 50 meses de pessoal). Para projetos menores, deve-se usar um expoente de $1/2$ [6].

Essa fórmula pode ser usada para estimar projetos de pequeno à grande porte [2].

### 5.3.3 Relações Lineares

Uma vez determinado o esforço em horas, necessário para a realização de uma atividade, para obter sua estimativa de duração, basta dividir esse valor pelo número de horas trabalhadas pela equipe alocada.

**Equação 32**

$$Prazo = \frac{Esforço}{Resursos}$$

Ou seja, o prazo previsto será a razão entre o esforço estimado e a quantidade de recursos alocados na execução do projeto.

### 5.3.4 Os Problemas de Encurtar o Cronograma

Após o cronograma normal ter sido calculado, a questão que surge muitas vezes é, "Quanto pode-se encurtar o cronograma se for necessário?" A resposta depende de se o conjunto de recursos é flexível. Se os recursos podem ser cortados, o cronograma pode ser reduzido tanto quanto for necessário. Isso equivale a fazer menos tarefas em menos tempo.

Se o conjunto de recursos não é flexível, encurtar o cronograma depende de mais pessoal para fazer mais trabalho em menos tempo, que é razoável, até certo ponto.

Ao longo das últimas décadas muitos pesquisadores de estimativas têm investigado os efeitos de não se cumprir o cronograma normal. A figura 16 sumariza os resultados.

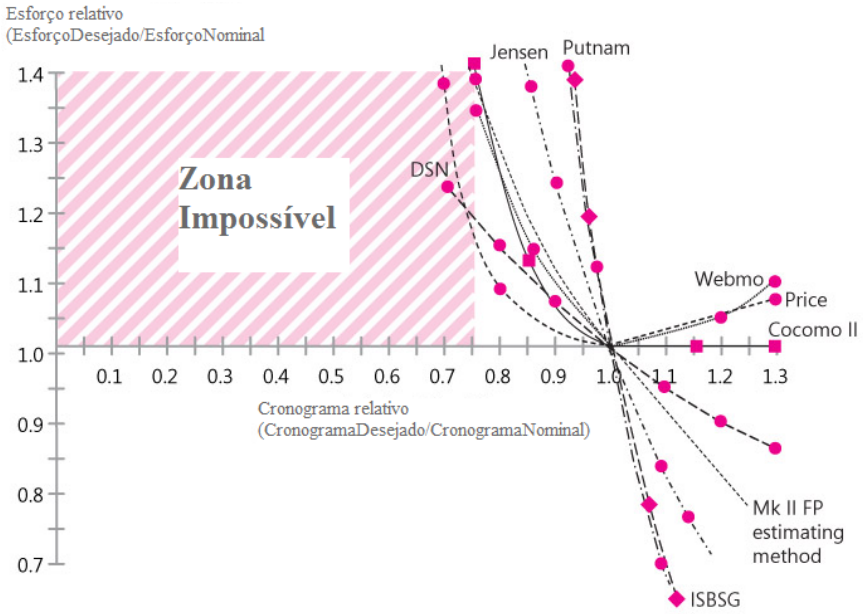


**Figura 16 - Efeitos da compressão de cronograma. Adaptado de [2].**

O eixo horizontal representa a relação entre o cronograma normal e o comprimido, Onde o valor 1 representa o cronograma normal. O eixo vertical representa o esforço total necessário quando o cronograma é comprimido.

Observando a figura pode-se concluir que encurtando o cronograma o esforço global cresce. O grau em que o esforço aumenta quando se encurta o cronograma varia para cada pesquisador, mas todos concordam que isso acontece.

Como pode ser visto na figura há uma zona de alcance impossível. Ou seja, não há como encurtar o cronograma além dessa zona. Encontrar os limites dessa zona não é um problema fácil, mas o consenso entre os pesquisadores é que uma compressão de mais de 25% no cronograma normal não é possível. Ele simplesmente não pode ser feito [18] [25].

# 6 PROPOSTA DE UM MODELO PARA A GESTÃO DE ESTIMATIVAS DE SOFTWARE DA FÁBRICA DE TIC GAIA.

Baseado nos estudos feitos no decorrer deste trabalho e apoiado pelo modelo do PMBOK [46] no que diz respeito à gestão do processo de estimativas foi proposto um modelo para gerir as estimativas da fábrica de TIC GAIA com intuito de agregar mais qualidade ao processo de desenvolvimento, dando um suporte maior a gerencia de projetos em especial a área de planejamento e controle do projeto.

Processo é o conjunto de atividades inter-relacionadas, que transforma entradas em saídas. O processo de gestão de estimativas tem como objetivo indicar ao gerente uma melhor forma de estimar, revisar e recalcular o tamanho, esforço, tempo e custos à medida que novos artefatos estejam disponíveis durante o desenvolvimento do projeto, visando subsidiar decisões gerenciais ao longo da gestão do projeto.

## 6.1 ELEMENTOS COMUNS DE UM PROCEDIMENTO PADRONIZADO DE ESTIMATIVA

Segundo MacConnell [2] um processo padronizado de estimativas normalmente deve:

- Enfatizar a contagem e computação quando possível, ao invés de usar o julgamento;
- Usar múltiplas abordagens e comparar os resultados;
- Ter pontos de reestimativa no projeto;
- Conter uma descrição clara da inexatidão das estimativas;
- Definir quando a estimativa pode ser usada como base para compromissos internos e externos.
- Arquivar os dados das estimativas e rever a eficácia do procedimento.

Para o procedimento padronizado de estimativa funcionar, é importante que a organização trate o procedimento como um padrão. Desvios do procedimento devem ser justificados por escrito, e eles devem ser raros.

O procedimento pode ser alterado no final de um projeto, motivado por um desejo de melhorar a precisão do processo para projetos futuros. O procedimento não deve ser alterado “em andamento”. Tais mudanças são muito propensas ao viés que irá prejudicar tanto a precisão da estimativa específica em questão quanto à eficácia do procedimento para projetos futuros.

Segundo MacConnell [2] o uso de ciclos de vida no desenvolvimento de software ajuda no processo de estimativas, principalmente porque as entre fases do ciclo são pontos propícios de reestimativa. Os ciclos de vida identificam atividades de desenvolvimento que normalmente são executadas em cada estágio. Ele também determina os critérios de saída que definem se o projeto pode prosseguir para a próxima fase.

O modelo será proposto em cima do ciclo de vida que a GAIA usa e terá como base o processo padrão de estimativa [43]. O processo de desenvolvimento da GAIA pode ser visto na figura 17 e sua descrição encontra-se no anexo A.

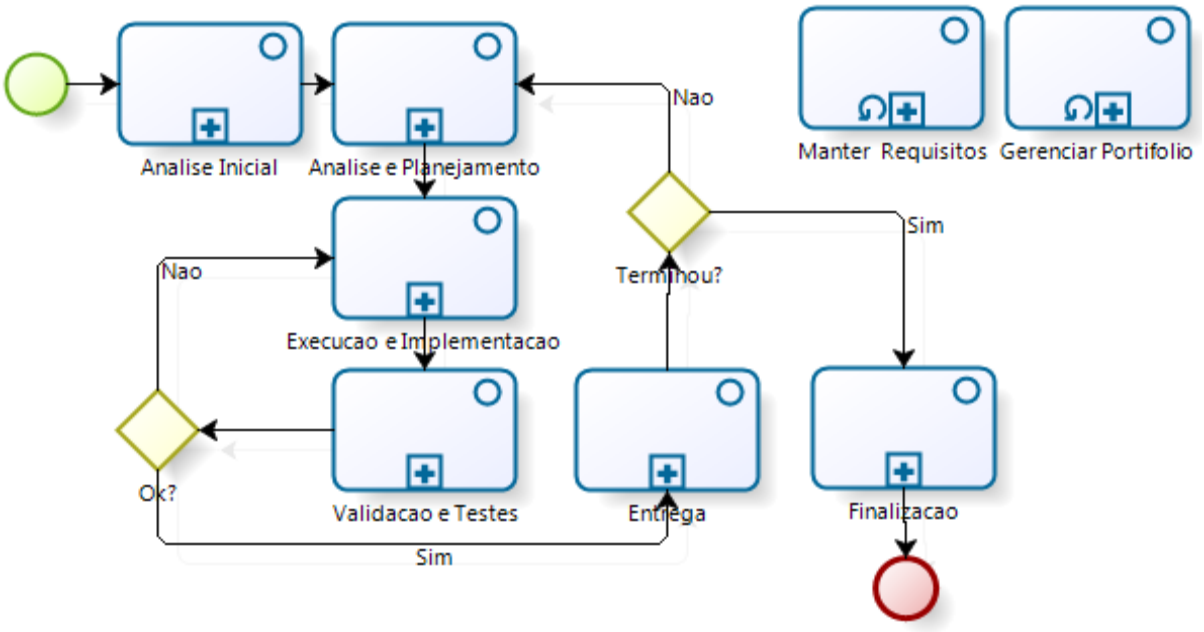


**Figura 17 - Processo de desenvolvimento GAIA**

## 6.2 MODELO PADRÃO DE GESTÃO

Com base nesses dados foi proposto o seguinte modelo, presente na tabela 11, para gerenciar as estimativas da fábrica de TIC GAIA:

**I. Análise Inicial:**

a. Criar pelo menos uma estimativa de prazo usando as seguintes abordagens:
   1. Uma pessoa estima usando *bottom-up* através de uma WBS;
   2. Outra pessoa *top-down* usando analogia com projetos passados semelhantes;
   3. E uma terceira usando o PERT.
b. Usar *Wideband Delphi* para convergir às estimativas em um valor único.
c. Estimativa deverá ser apresentada internamente com uma faixa de -50% e + 100%, ou usando a variância das estimativas (do banco de dados) dessa fase.
d. Armazenar as estimativas desta fase.

**II. Análise e Planejamento:**

a. Reestimar usando as abordagens 1 e 2 do passo I(a).
b. Criar estimativas de tamanho usando PF.
   1.Computar os pontos de funções usando a especificação de requisitos (usar NESMA caso a especificação for inicial).
   2. Computar UCP.
      - Projetar os UCP em PF através de uma equação que relacione as duas medidas.
   3. Usar a média das estimativas 1 e 2 e calibrar com os dados históricos.
   4.Estimar o esforço e o cronograma dependendo do tamanho do projeto
      - Pequeno: usar relações lineares pode-se usar a equação de Roetzheim para estimar o cronograma caso haja dados históricos.
      - Médio: Usar a equação de esforço do COCOMO II (calibrado com a organização) e a equação básica do cronograma, ou equação de Roetzheim (ter dados históricos).
      - Grande: Usar a equação de esforço do COCOMO II (calibrado), a equação de cronograma do COCOMO II e a equação básica e analisar a convergência. (Pode-se usar o Modelo de Putnam caso ele se adapte melhor, mas é necessário testes para isso).
c. Iterar II(a) e II(b) até que as estimativas convirjam dentro de uma faixa de ±5% para cada estimativa. Usar a média para achar o valor nominal, N.
d. Mostrar a estimativa em uma faixa de 0.8N a 1.25N ou usando a variância das estimativas (do banco de dados) dessa fase.

e. Armazenar as estimativas dessa fase.

## III. Execução e Implementação:

a. Construir uma estimativa *bottom-up*.
   1. Criar uma lista detalhada de tarefas.
   2. Cada desenvolvedor, *tester* ou qualquer um que contribua para o esforço do projeto deve estimar o seu trabalho (módulo) em pior caso, caso médio e melhor caso.
   3. O valor nominal é computado usando PERT.

b. Comparar as estimativas II(d) e III(a) e computar o valor nominal pela formula: $\frac{(2*Estimativa_{maior}+Estimativa_{menor})}{3}$

c. Apresentar as estimativas em uma faixa de 1N e 1.1N ou usando a variância das estimativas (do banco de dados) dessa fase.

d. Nessa fase podem ser estabelecidos compromissos externos

e. Armazenar Estimativas dessa fase.

## IV. Validação e Teste

a. Nessa fase podem ser estabelecidos compromissos externos.

b. Comparar resultados das estimativas com o resultado atual, pela fórmula: $Esforço_{restante} = \frac{Esforço_{planejadoRestante}}{(\frac{Esforço_{atualParaData}}{Esforço_{planejadoParaData}})}$, Pode ser estabelecida alguma medida de relação entre o esforço restante e o esforço restante planejado, como a média, por exemplo, e esse ser o novo esforço nominal.

c. As estimativas podem ser apresentadas em uma faixa de 0.9N e 1.1N ou usando a variância das estimativas (do banco de dados) dessa fase.

d. Armazenar as estimativas dessa fase.

## V. Entrega

a. Projeto deve ser reestimado a qualquer momento em resposta a importantes mudanças nas premissas de projeto, disparando a atividade II, e calibrando com os dados da atividade III e atualizando os dados da atividade IV.

## VI. Finalização

a. Coletar a arquivar os dados do projeto atual.

b. Rever a precisão de cada estimativa.
   1. Analisar a causa de eventuais erros e imprecisões.
   2. Avaliar se a mesma precisão poderia ser alcançada com menos esforço.
   3. Propor melhorias ou revisões do modelo padrão.

**Tabela 11 - Modelo Padrão Proposto.**

## 6.3 COMENTÁRIOS SOBRE O MODELO

Nesta seção serão discutidas e comentadas cada uma das fases do modelo proposto.

### 6.3.1 Análise Inicial

A intenção dessa fase do modelo é prover uma estimativa rápida baseada nos artefatos produzidos na definição do escopo. Essa estimativa utiliza-se de artefatos produzidos pela fase de análise inicial do processo de desenvolvimento como, por exemplo, a WBS do projeto.

São utilizadas três abordagens diferentes para se estimar o cronograma: *bottom-up* (seção 3.6.1), julgamento de especialistas estruturado PERT (seção 3.4.1) e *top-down* combinado com analogia (seção 3.6.2 e 3.7 respectivamente). Como essa fase do processo de desenvolvimento ainda há muitas incertezas acerca do projeto a ser desenvolvido, é interessante usar metodologias diferentes para se estimar, pois cada tipo de metodologia pode ser afetada de uma forma diferente pelas incertezas do projeto e na hora de criar uma estimativa final convém descartar as áreas de uma metodologia que não são muito precisas e usar as outra metodologia.

As metodologias foram escolhidas com base nos artefatos e dados já disponíveis nessa fase do processo de desenvolvimento como observado anteriormente. O *bottom-up* pode ser utilizado já nessa fase que tem como artefato uma WBS do projeto. Vale destacar que será necessário reestimar sempre que essa WBS for modificada, o que acontece corriqueiramente em todo processo de desenvolvimento. *Top-down* combinado com analogia foi empregado nessa etapa porque as características gerais do projeto já foram acordadas com o cliente na análise inicial e o banco de dados de projetos semelhantes é algo disponível desde o início. É válido recorrer também ao julgamento de especialistas devido ao grau de abstração que o projeto tem, pois a experiência de um estimador ou gerente pode ser útil, mas como visto no decorrer do texto essa forma de estimar é muito imprecisa, então é usado um método estruturado (PERT) para agregar qualidade a essa estimativa tentando eliminar os preceitos que rondam o julgamento.

Cada metodologia é estimada por um estimador diferente para que uma estimativa não influencie na outra.

Depois de concluídas as três estimativas é utilizada a técnica de *Wideband Delphi* (seção 3.7.2) que é uma técnicas de julgamento em grupos estruturada para convergir as estimativas em uma de valor único. Essa técnica gera um bom percentual de precisão para criar um valor único a partir de várias estimativas, principalmente quando não se conhece muito do sistema a ser confeccionado como é o caso de projetos na fase de análise inicial.

Após isso a estimativa é apresentada e armazenada. O interessante aqui é apresenta-la com uma variância que pode ser pré-definida para a fase ou baseada na variância da estimativa nessa fase comparada a de projetos já finalizados usando dados históricos caso estejam disponíveis.

A figura 18 ilustra o dinamismo dessa fase.

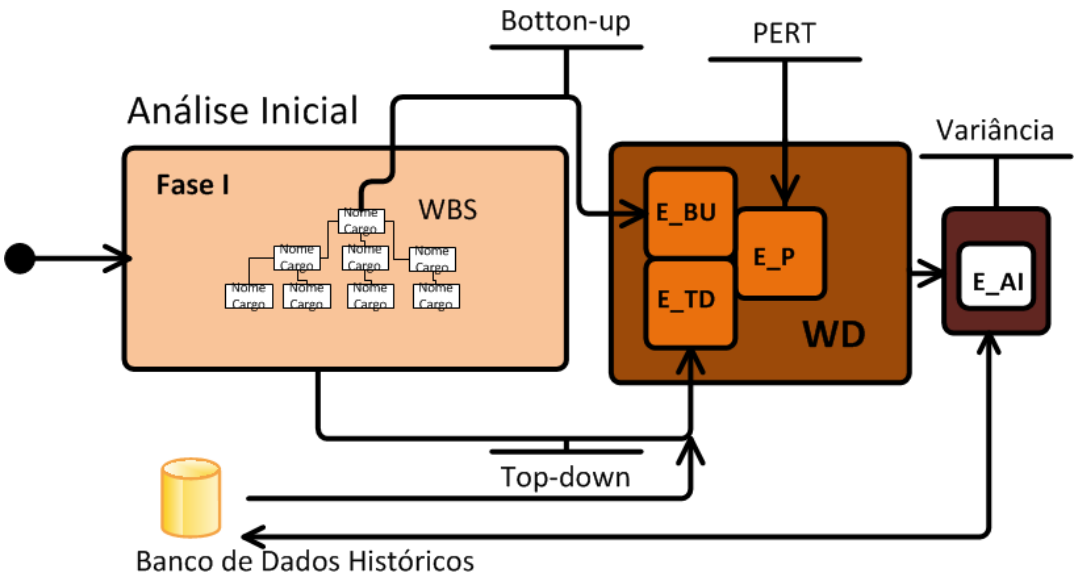


**Figura 18 – Processo de estimativa da fase de Análise Inicial.**

Como pode ser observado pela figura 18 a fase de análise inicial do processo de desenvolvimento gera uma WBS que é usada para criar a estimativa *bottom-up*, a estimativa *top-down* é feita usando as características gerais do projeto provenientes da análise inicial que também usa o banco de dados históricos como fonte de analogia. Por fim a estimativa PERT é computada e a técnica WD (*Wideband Delphi)* é usada para convergir as estimativas que são apresentadas segundo sua variância e armazenadas no banco de dados históricos.

### 6.3.2 Análise e Planejamento

O intuito dessa etapa é criar uma estimativa que servirá de base para o projeto inteiro. Como nessa fase do processo de desenvolvimento a grande maioria de documentos sobre o projeto já extaram disponíveis as estimativas tendem a ser mais precisas, principalmente porque elas se apoiaram nesses artefatos que são mais concretos do que os da fase anterior.

O projeto é reestimado usando as abordagens *bottom-up* e *top-down* com analogia, pois nessa altura os dados do projeto com certeza mudaram ou estão em um nível maior de detalhamento fato esse que justifica a reestimativa.

Nessa fase a especificação de requisitos já está disponível, então ela é usada para estimar o tamanho do software (em pontos de função em razão dos fatos comentados na seção 5.1 a 5.1.2) essa estimativa serve de parâmetro de entrada para as estimativas de esforço e cronograma. Esse tipo de abordagem gera estimativas mais precisas por se basear em fatos reais do projeto [2].

O tamanho em pontos de função é estimado usando APF com base na especificação de requisitos (seção 5.1.3), e caso a especificação for inicial utiliza-se a técnica de contagem indicativa NESMA (seção 5.1.4). Ainda pode-se optar em usar UCP (seção 5.1.5) como medida, nesse caso a contagem do tamanho é feita em cima do modelo de casos de uso do projeto. É válido também para tentar aumentar a precisão (também aumenta o esforço para se criar a estimativa) estimar usando as duas abordagens (APF e UCP) e depois projetar os UCPs em PF utilizando alguma correlação com base nos projetos passados (pode ser utilizado regressão linear) e calcular a média das duas estimativas para ser o valor nominal.

Após isso o esforço e o cronograma são estimados dependendo do tamanho do projeto (seção 3.2). Os projetos desenvolvidos pela GAIA geralmente são de tamanho pequeno, diante disso o mais recomendado é a utilização de relações lineares, como taxa de entrega e produtividade (seção 5.2.3), para se estimar o esforço. Esse método requer menos tempo para se estimar e como projetos pequenos não são afetados pela desconomia de escala (seção 2.9.1) esse efeito não precisa ser calculado e levado em consideração o que viabiliza o uso de tal estratégia. O cronograma inicialmente pode ser estimado também por relações lineares (seção 5.3.3) quando não há dados históricos disponíveis. O que acontece é que a relações lineares não traduzem bem o cronograma [43]. Então quando os dados históricos

estão disponíveis faz-se o uso de outra abordagem sendo a equação de Roetzheim a escolhida (seção 5.3.2).

Os projetos de médio e grande porte já se utilizam de modelos mais formais para se estimar o esforço e cronograma, o que requer um grau maior de maturidade da organização principalmente para que ela consiga concluir com sucesso projetos desse porte. No caso do modelo de gestão é utilizado o COCOMO II (seção 4.1) para se estimar o esforço e para o cronograma as equações das seções 5.3.1 e 5.3.2. Ainda pode ser feito testes com outros modelos paramétricos, como o Putnam SLIM, para avaliar sua precisão e dependendo dos resultados incorporá-los ao procedimento padrão.

Feito isso as estimativas criadas nessa fase são iteradas a fim de que seus valores convirjam para algum percentual aceitável pela organização. Depois desse processo encontra-se a média e a apresenta como valor nominal respaldado pela variância dessa fase como na análise inicial. Ao passar para a próxima fase as estimativas dessa etapa são armazenadas.

A figura 19 mostra um esquema de como essa etapa funciona.

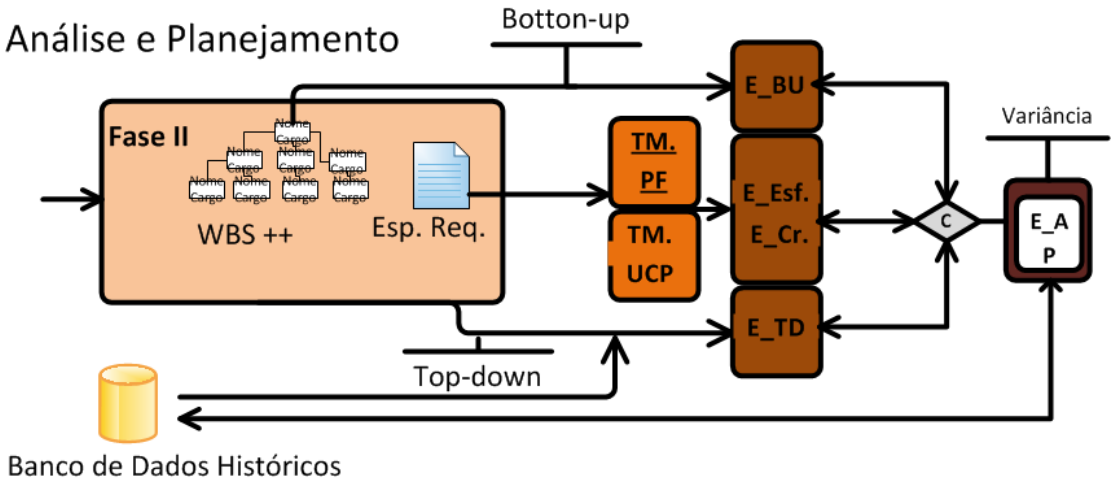


**Figura 19 - Processo de estimativa da fase de Análise e Planejamento.**

Segundo a ilustração nessa fase há uma WBS mais detalhada a partir daí a estimativa *bottom-up* é refeita. Possivelmente as características do projeto também sofreram alterações então o projeto é reestimado usando *top-down* combinado com analogia. A análise e planejamento prove a especificação de requisitos que é usada para estimar o tamanho em PF ou UCP e usando essa medida de tamanho como entrada o esforço e o cronograma são

estimados. Após isso as três estimativas são iteradas para que convirjam para uma faixa aceitável, posteriormente a estimativa é apresentada e armazenada como na fase anterior.

### 6.3.3 Execução e Implementação

Essa fase tem o objetivo de calibrar a estimativa da fase anterior com a estimativa do trabalho que realmente será efetuado.

Na fase de execução e implementação do processo de desenvolvimento já está disponível uma especificação detalhada do projeto, e através dessa é criada uma WBS com esse nível de detalhe de todas as funcionalidades e seus respectivos encarregados. Com esse nível de especificidade cada desenvolvedor, *tester* ou outro membro que contribua para o esforço do projeto já tem suas tarefas definidas. Então cada um estima o seu trabalho usando o PERT, pois como observado na seção 2.8.4 os desenvolvedores tendem estimar com boa precisão o seu trabalho.

Essa estimativa é comparada com a da fase anterior e o valor nominal é calculado por uma fórmula ponderada, onde é atribuído peso 2 a estimativa de maior valor pois sua penalidade é menor do que subestimar como visto na seção 2.7.

Por fim as estimativas são apresentadas em uma faixa pré-definida ou por sua variância igual nas fases anteriores, e essas também são armazenadas.

A figura 20 demostra como o processo funciona.

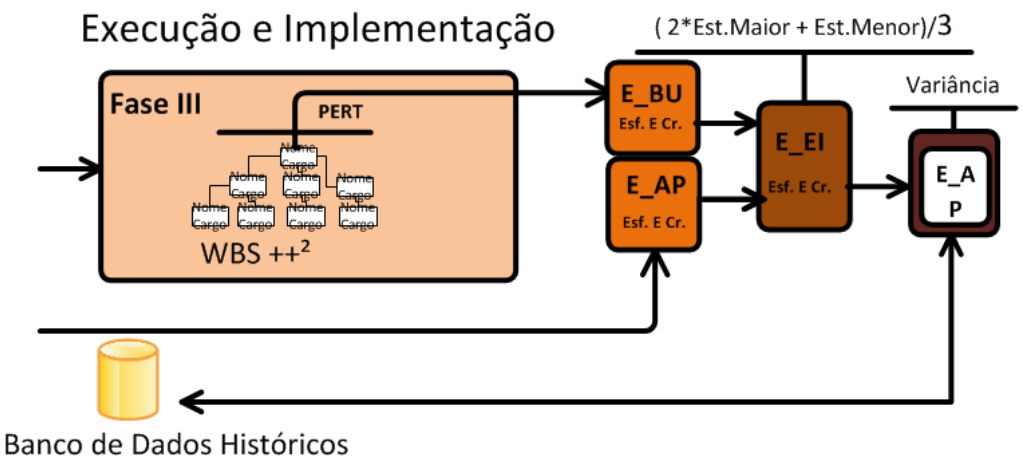


**Figura 20 - Processo de estimativa da fase de Execução e Implementação.**

Observando a figura 20 nota-se que nessa fase já se obtém uma WBS bem detalhada e através dela a estimativa *bottom-up* é refeita só que agora por todos os membros da equipe, usando PERT para calcular o valor nominal de cada funcionalidade. Essa estimativa é comparada com a E_AP (estimativa da Análise Inicial) e a E_EI (estimativa de Execução e Implementação) é calculada e apresentada com sua variância e depois armazenada.

### 6.3.4 Validação e Teste

Na validação e teste as estimativas são atualizadas e revisadas.

Para isso é feita a comparação das estimativas pela fórmula presente na tabela 11 na parte IV, onde o esforço restante é calculado usando o esforço planejado restante, o esforço atual e o esforço planejado para a data. Um novo valor para o esforço pode ser obtido através de uma correlação, como a média, entre o esforço obtido nessa fase e o esforço restante planejado. O cronograma deve ser atualizado caso haja mudanças nos valores do esforço restante do projeto.

E como nos itens anteriores as estimativas são apresentadas em uma faixa pré-definida ou usando a variância média dessa fase e finalmente são armazenadas.

A figura 21 traz a dinâmica dessa fase.

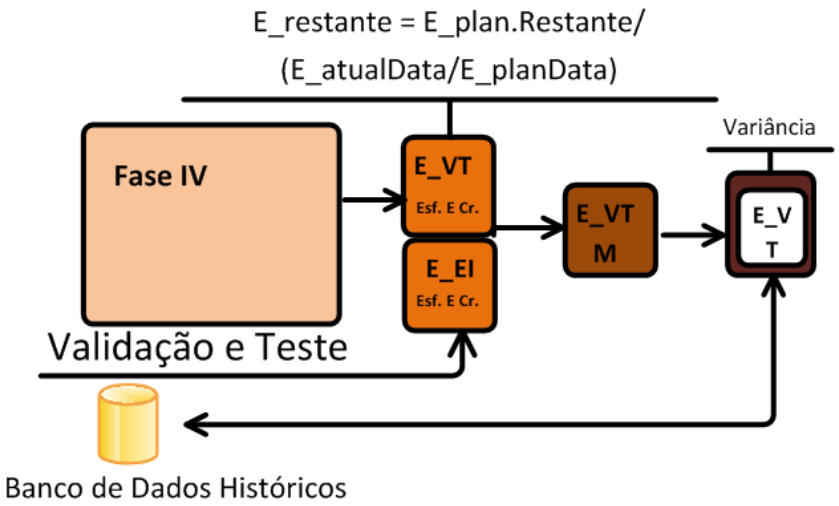


**Figura 21 - Processo de estimativa da fase de Validação e Teste.**

Como consta na figura 21 a E_VT (estimativa de Validação e Teste) é calculada e comparada com a E_EI a partir daí uma estimativa nova é criada usando o valor médio entre as duas por exemplo. Essa é apresentada e armazenada conforme as fases anteriores.

### 6.3.5 Entrega

Etapa de validação com o cliente, onde ele verifica se o sistema está de acordo com os requisitos. Caso o projeto esteja de acordo ele parte para a fase de finalização. Se o projeto não foi aprovado, a atividade de análise e planejamento é ativada novamente no processo de desenvolvimento e consequentemente é disparada também suas práticas de estimativas para essa fase de acordo com o modelo padrão proposto e conforme o projeto caminhar novamente pelas fases anteriores suas práticas de estimativas também são reativadas.

### 6.3.6 Finalização

Aqui os dados do projeto são armazenados. Os valores de tamanho, esforço e cronograma são computados e armazenados com o intuito de servir de base para estimar projetos futuros. Esses dados serão utilizados em todas as outras fases do modelo como visto anteriormente. Esses valores servem de parâmetros das estimativas *top-down* com analogia, para gerar a variância média de cada fase do modelo, são usados toda vez que o modelo necessitar ser calibrado com os dados da organização e ainda são úteis como meio de calibração dos modelos formais como o COCOMO II.

Na finalização a precisão do modelo também é revisada com a intenção de encontrar eventuais erros de precisão, otimizar o modelo e propor melhorias, conforme a figura 22.

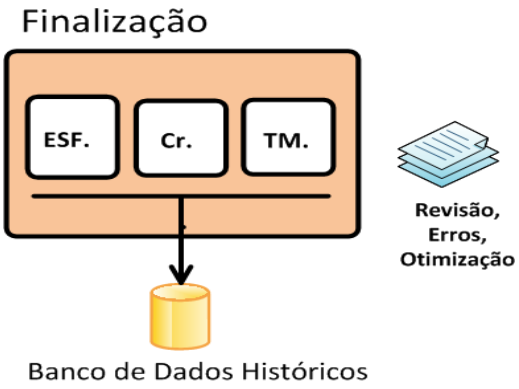


**Figura 22 - Processo de estimativa da fase de Finalização.**

A figura 22 mostra que os valores de esforço, tamanho e cronograma são computados e armazenados no banco de dados históricos. E o modelo é revisado.

### 6.3.7 Padronização do Procedimento

O modelo acimo exposto aborda cada elemento comumente encontrado em um processo padronizado de estimativa:

**Enfatizar a contagem e computação quando possível, invés de usar o julgamento**. Na fase de análise inicial são feitas estimativas usando *bottom-up* e *top-down* através da WBS, PERT e analogia. Nenhuma dessas abordagens principalmente nessa fase é muito precisa, mas envolve alguma computação ao invés de puro julgamento.

**Usar múltiplas abordagens e comparar os resultados**. As três primeiras estimativas já usam abordagem múltipla e comparação dos resultados. Isso acontece mais vezes no decorrer do modelo.

**Ter pontos de reestimativa no projeto.** A reestimativa é frequente no projeto, só em cada marco há uma reestimativa, sem contar nas reestimativas dentro de cada fase, como na fase II.

**Conter uma descrição clara da inexatidão das estimativas**. Cada fase do processo apresenta a faixa de possível variação da estimativa como, por exemplo, na fase I a faixa é de -50% a +100% estreitando essa faixa durante o modelo para ±10%.

**Definir quando a estimativa pode ser usada como base para compromissos internos e externos**. Isso é indicado na fase IV.

**Arquivar os dados das estimativas e rever a eficácia do procedimento**. A fase VI é encarregada dessa atividade. Mas em cada fase são coletados dados das estimativas para serem usados em projetos futuros comparando não só com o resultado do projeto inteiro, mas também com cada fase do processo.

### 6.3.8 Melhorias no Processo

MacConnell [2] ainda aponta algumas perguntas que podem ser feitas ao final do projeto com intuito de melhorar o modelo padrão de estimativas, são elas:

- As estimativas foram precisas? O resultado final estava dentro do intervalo?
- As faixas foram largas o suficiente? Poderiam ser feitas mais estreitas e ainda daria conta da variabilidade observada?
- A tendência foi subestimar, superestimar ou foi neutro?
- As fontes de incerteza estavam presentes?
- Quais as técnicas produziram estimativas mais precisas? Será que essas técnicas geralmente produzem as estimativas mais precisas, ou que elas simplesmente produziram as melhores estimativas neste caso?
- As reestimativas foram no momento certo? E o número de vezes foi: alto, baixo ou neutro?
- O processo de estimativa está mais elaborado do que precisa ser? Como poderia agilizá-lo sem perder a precisão?

# 7 CONCLUSÃO

No decorrer desse trabalho foram abordados diversos conceitos sobre estimativas, seu relacionamento com outras áreas do desenvolvimento, suas influências, fontes de erros além de explorar os modelos mais conhecidos, metodologias e técnicas, provando que o esse campo de estudo é vasto.

Mesmo com todas essas áreas em estudo observou-se que processo de estimar é muito subjetivo, devido a isso não há ainda um modelo padrão, não há um consenso em nenhuma das subáreas estudadas, fato esse que vem instigando ainda mais estudos e esforços para resolver esse dilema onde uma possível solução poderia ser usada não só em estimativas de software, mas sim em diversas outras áreas do conhecimento que sofrem da mesma subjetividade.

Pelos conceitos apresentados no capítulo 2 observou-se um forte relacionamento entre estimativas e o planejamento, onde se pode concluir que as estimativas são a base para um planejamento efetivo que por sua vez subsidia o sucesso do projeto.

Outro fator importante observado foi a deseconomia de escala que eventualmente atinge os projetos de software à medida que seu tamanho cresce, essa é uma das principais influências na estimativa junto com o tipo de software que será desenvolvido e os fatores pessoais.

Diante das metodologias apresentadas foi notado um consenso entre os autores de que a metodologia de julgamento de especialistas apesar de ser a mais usada na prática deve ser evitada, pois produz os piores resultados. Uma alternativa criada foi o julgamento de especialistas estruturado com revisores de grupo em conjunto com a técnica *Widebrand Delphi* que melhora muito as estimativas comparado ao julgamento não estruturado.

A metodologia *bottom-up* é fortemente recomendada, pois obtém um bom nível de precisão, porém precisa-se de mais esforço e conhecimento do projeto para usá-la, uma boa alternativa é a metodologia *top-down* combinada com analogia que produzem resultados bons com um esforço menor.

Os modelos paramétricos apresentados são os mais comuns e usados na prática, dentre eles merece destaque o modelo COCOMO II que é fruto de um rigoroso estudo estatísticos e fornece boas fórmulas de esforço e cronograma juntamente com seus

direcionadores de custo e fatores de escala que ajudam a diminuir e identificar o foco das influências na estimativa, como a deseconomia de escala.

Dentre as técnicas apresentadas para estimar o tamanho foram exploradas as mais proeminentes que são APF e UCP, APF já é um padrão aceito pelo mercado e é ativamente melhorado, o UCP vem crescendo e melhorando além de ter a vantagem de poder ser estimado antes da APF e facilitar a automatização da estimativa. As estimativas de esforço e cronograma seguem basicamente fórmulas padronizadas, relações lineares que apresentam bom rendimento em projetos pequenos. Vale destacar a equação básica do cronograma que é praticamente aceita pela maioria dos autores.

Todos os modelos, metodologias e técnicas estudadas têm pontos fortes e fracos e devem ser usados principalmente em conjunto para aumentar a precisão e em um cenário propício, esse foi um dos principais objetivos da criação do modelo.

O modelo proposto cobriu praticamente todas as áreas estudadas, tentando captar o que cada técnica/metodologia tem de melhor em cada tempo, para que ele possa subsidiar estimativas de precisão, mitigando as fontes de erros e incertezas que rondam esse processo. Esse modelo ainda precisa ser testado e a partir dos testes revisado, se necessário, e principalmente ser continuamente melhorado. Dessa forma, ele irá agregar maior qualidade no processo de desenvolvimento dando um maior suporte a gerência de projetos.

# 8 REFERÊNCIAS

[1] CHEMUTURI, M. K. *Software Estimation Best Practices, Tools & Techniques: A Complete Guide for Software Projects Estimators*. 1.ed.: J. Ross Publishing, 2009. 320p.

[2] MCCONNELL, S. *Software Estimation: Demystifying the Black Art*. 1.ed. Washington: Microssoft Press, 2006. 308p.

[3] BUNDSCHUH, M.; DEKKERS, C. *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement*. 1.ed. Berlin: Springer, 2008. 680p.

[4] CAPERS, J. *Estimating Software Costs*: *Bringing Realism to Estimating.* 2.ed. New York: McGraw-Hill, 2007. 644p.

[5] CONTE, D. S.; DUNSMORE H. E.; SHEN, V. Y. *Software Engineering Metrics and Model*. 1.ed. Menlo Park: Benjamin/Cummings, 1986. 403p.

[6] STUTZKE, D. R. *Estimating Software-Intensive Systems: Projects, Products, and Processes*. Upper Saddle River: Addison-Wesley, 2005. 944p.

[7] BOEHM, W. B. *Software Engineering Economics*. 1.ed. Englewood Cliffs: Prentice Hall, 1981. 767p.

[8] GOLDRATT, E. M. *Critical Chain*. 1.ed. Great Barrington: The North River Press, 1997. 246p.

[9] LAWLIS P. K.; FLOWE, R. M.; THORDAHL, J. B. *A Correlational Study of the CMM and Software Development Performance.* Crosstalk, p. 21-25, 1995.

[10] ISBSG. *Practical Project Estimation, 2nd Edition: A Toolkit for Estimating Software Development Effort and Duration*. 2.ed. Australia: International Software Benchmarking Standards Group, 2005. 124p.

[11] Standish Group, *The CHAOS Report*. 2003.

[12] BOEHM, B. W.; TURNER R. *Balancing Agility and Discipline: A Guide for the Perplexed.* 1.ed. Boston: Addison-Wesley, 2004. 266p.

[13] MCCONNELL, S. *Code Complete*, 2.ed. Redmond :Microsoft Press, 2004. 928p.

[14] GLASS, R. L. *IS Field: Stress Up, Satisfaction Down.* Software Practitioner, p. 1-3, 1994.

[15] CAPERS, J. *Assessment and Control of Software Risks*. 1.ed. Englewood Cliffs: Prentice Hall, 1994. 464p.

[16] LAIRD. L. M. *The Limitations of Estimation.* IT PRO, Dez. 2006, p. 40-45.

[17] JØRGENSEN, M. *A Review of Studies on Expert Estimation of Software Development Effort*. Journal of Systems and Software 70, 2004, p. 37-60.

[18] BOEHM, B. W.; ABTS, C.; BROWN, A. W.; CHULANI, S.; CLARK, B. K. HOROWITZ, E.; MADACHY, R.; REIFER, D.; STEECE, B. *Software Cost Estimation with COCOMO II*. 1.ed. Upper Saddle River: Prentice Hall PTR, 2000. 544p.

[19] COOMBS, P. *IT Project Estimation: A Practical Guide to the Costing of Software*. 1.ed. Cambridge: Cambridge Univeristy Press, 2003. 184p.

[20] PRECHELT, L. *An Empirical Comparison of Seven Programming Languages.* IEEE Computer, Out. 2000. p. 23-29.

[21] DEMARCO, T.; LISTER, T. *Peopleware: Productive Projects and Teams*. 2.ed. New York: Dorset House, 1999. 245p.

[22] KITCHENHAM, B.; PFLEEGER, S. L.; MCCOLL, B.; EAGAN, S. *A Case Study of Maintenance Estimation Accuracy*, Journal of Systems and Software, 2002.

[23] TODD, P.; BENBASAT, I. *Inducing Compensatory Information Processing Through Decision Aids That Facilitate Effort Reduction: an Experimental Assessment.* Journal of Behavioral Decision Making 13(1), 2000, p. 91-106.

[24] LEDERE, A. L.; PRASAD, J. *Nine Management Guideliness for Better Cost Estimating*. Communications of the ACM, Fev. 1992, p. 51-59.

[25] PUTNAM, L. H.; MYERS, W. *Five Core Metrics: The Intelligence Behind Successful Software Manageme*. New York: Dorset House, 2003. 328p.

[26] FOSS, T.; STENSRUD, E.; KICHENHAM, B.; MVRTVEIT, I. *A Simulation Study of the Model Evaluation Criterion MMRE.* Journal IEEE Transactions on Software Engineering, v. 29 n. 11, Nov. 2003

[27] TOCKEY, S. *Return on Software*. 1.ed. Boston: Addison-Wesley, 2005. 621p.

[28] PFLEEGER, S. L.; WU, F.; LEWIS, R. *Software Cost Estimation and Sizing Methods: Issues, and Guidelines*. Santa Monica: Rand Publishing, 2005. 72 p.

[29] JØRGENSEN, M. *Top-down and bottom-up expert estimation of software development effort*. Information And Software Technology, Amsterdã, v. 46, p.3-16, 1 jan. 2004.

[30] KEUNG, J. *Software Development Cost Estimation Using Analogy: A Review*, Software Engineering Conference, ASWEC '09. Australian , pp.327-336, April 2009 doi: 10.1109/ASWEC.2009.32

[31] SHEPPERD, M.; SCHOFIELD, C.; KITCHNHAM, B. *Effort estimation using analogy*. In International Conference on Software Engineeri ng. p. 170-178, Berlin, 1996.

[32] BRIAND, L. C.; EMAN, K. E.; MAXWELL, K. D. *An assessment and comparison of common software cost estimation modeling techniques*. In International Conference on Software Engineering, LA, 1999.

[33] SHEPPERD, M. *Case-based reasoning and software engineering*. Technical Report TR02-08, Bournemouth University, UK, 2002.

[34] WALKERDEN, F. ; JEFFERY, R. *An empirical study of analogy-basd software effort estimation*. Empirical Software Engineering, 4: p.135-158, 1999.

[35] ARMSTRONG, J. S. *Principles of forecasting: A handbook for researchers and practitioners*. 1.ed. Boston: Springer, 2001. 864p.

[36] GIOMBETTI M. *Cost / Benefit Aspects of Software Quality Assurance - Selected Topics in Software Quali*t. Report: TUM I0824: Institut für Informatik - Technische Universität München, 2008.

[37] PUTNAM, L. H. *A general empirical solution to the macro software sizing and estimating problem*, IEEE Trans. Software Eng., vol. 4, 1978, pp. 345-361.

[38] GIOMBETTI M. *An Analysis of Software Cost Estimation Methods with Regard to Quality Requirements*. 2010. 160p. Dissertação de Mestrado. - Fakultät für Informatik.

[39] FISCHMAN, L.; MCRITCHIE, K.; GALORATH D. D. *Inside SEER-SEM*, CrossTalk, The Journal of Defense Software Engineering (April 2005), 2005.

[40] ISBSG.; HILL, P .R. *Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration*. 1.ed.:McGraw-Hill, 2010. 312p.

[41] ALBRECHT, A. J. *Measuring Application Development Productivity*, Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, Out. 1979, p. 83-92.

[42] LONGSTREET, D. *Function Point Analysis Training Course*. Longstreet Consultanting Inc, 2004.

[43] VAZQUEZ C. E.; SIMÕES, G. S.; AlBERT R. M., *Analise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software*. 6. ed, São Pualo: Érica, 2007.

[44] NESMA, *Early FPA Counting*. Disponível em http://www.nesma.nl/section/fpa/earlyfpa.htm. Acesso em Novembro de 2011.

[45] ANSELMO, F. *Métricas para Desenvolvedores*. 1.ed. Florianópolis: VisualBooks Editora, 2010. 186p.

[46] PMI, *Project Management Institute; Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK)*, 3.ed, Newtown square : Project Management Institute, 2008. 337p.

# ANEXOS

# ANEXO A – DESCRIÇÃO DO PROCESSO DE DESENVOLVIMENTO DA FÁBRICA DE TIC GAIA

O processo de desenvolvimento de software compreende um conjunto de atividades que engloba métodos, ferramentas e procedimentos, com o objetivo de produzir softwares que atendam aos requisitos especificados pelos usuários ou clientes.

A partir de uma perspectiva de gerenciamento baseada no PMBOK[46], o Processo de Desenvolvimento de Software da Fábrica de TIC GAIA é dividido em oito macro atividades cada uma concluída por um marco principal, a figura 17 apresenta esse processo. As oito macro atividades são: análise inicial, análise e planejamento, execução e implementação, validação e testes, entrega, finalização, manter requisitos e gerenciar portfólio. Cada macro atividade do processo é composto por atividades, sendo que cada uma destas atividades são descritas por um fluxo de trabalho composto por tarefas a serem realizadas pelos papéis do processo, gerando artefatos (atas, documentos, código fonte, planos, *checklists* entre outros).

Esse processo de desenvolvimento foi criado para que de forma gradual atinja o nível F de maturidade do MPS.br.

**Análise Inicial**: reunião com o cliente para entendimento do problema e definição do escopo. A GAIA investe na qualidade deste escopo, minimizando problemas de falta de entendimento, insatisfações futuras do cliente pelo fato do sistema não atender suas necessidades, evitando com isso o retrabalho. O resultado deste investimento é a minimização dos riscos do projeto. Para cada reunião é gerada uma ata que deve ser assinada por todos os participantes, firmando o comprometimento de todos os envolvidos e para que os assuntos tratados sejam disponibilizados eletronicamente a todos os demais integrantes do desenvolvimento deste produto. Ao término desta etapa, tem-se uma proposta para o cliente, incluindo o escopo que é representado por uma *Work Breakdown Structure* (WBS), premissas, riscos, o prazo estimado (em meses) para o desenvolvimento e o custo do projeto. Para estabelecimento dos prazos e custos utiliza-se um banco de dados histórico do desempenho da equipe em projetos similares;

**Análise e Planejamento**: Após a aprovação da proposta, deve-se iniciar o planejamento do projeto, por meio do refinamento dos requisitos e de especificações, dos riscos e prioridades de desenvolvimento, da expansão da WBS, da alocação de pessoas, da elaboração do cronograma, do estabelecimento de pontos de controle, do número de iterações e de quais requisitos serão desenvolvidos em cada iteração. É gerado um artefato intitulado

Plano de Projeto. Vale ressaltar que esta macro atividade do processo de desenvolvimento da GAIA ocorre de maneira iterativa, ou seja, após a primeira iteração definida e iniciada, no término da mesma, caso o desenvolvimento deva continuar, esta macro atividade é disparada novamente. Nesta macro atividade também ocorre o estabelecimento do grau de severidade para a aprovação ou não dos resultados das atividades pelo projeto. O grau de rigorosidade implica diretamente no controle da qualidade do projeto, ou seja, quanto menor a grau de rigorosidade, mais rígido é o processo de garantia de qualidade do projeto;

**Execução e Implementação**: Nesta macro atividade ocorre a especificação e a implementação dos respectivos requisitos e testes unitários. A especificação de requisitos deve ser verificada e validada. Caso ocorra uma quantidade igual ou superior de não conformidades aceitáveis para o projeto em questão, a iteração deve ser cancelada e um novo planejamento deve ser estabelecido levando-se em consideração os atrasos e as consequências dos mesmos. Após uma análise do resultado dos testes, decide-se, baseado também no grau de rigorosidade, por corrigir as não conformidades encontradas e realizar novamente os testes e partirmos para a próxima macro atividade intitulada Entrega ou cancelarmos a iteração e voltarmos para a macro atividade de Análise e Planejamento;

**Validação e Testes**: Nesta macro atividade são realizados os testes de integração do sistema, no qual a parte implementada na interação presente é avaliada no que diz respeito à comunicação com as demais parte do sistema ou sistemas terceiros já implementados.

**Entrega**: Esta macro atividade está responsável por implementar o resultado da macro atividade no e entregá-lo ao cliente. Também nesta atividade são coletadas lições aprendidas e *feedback* do cliente com relação ao projeto. Se o projeto ainda não terminou, a macro atividade de Análise e Planejamento é iniciada novamente. Do contrário, a macro atividade de Finalização é iniciada;

**Finalização**: Nesta macro atividade é realizada uma reunião de término do projeto, na qual são levantadas as lições aprendidas, sendo as mesmas registradas em ata para futuras consultas e melhorias no processo de desenvolvimento. É gerado um documento indicando o recebimento do produto pelo cliente e o término do projeto.

**Manter Requisitos**: Esta macro atividade é realizada de modo assíncrono ao processo, sendo responsável por receber as alterações de requisitos, avaliá-las e aprová-las, garantindo assim a coerência dos requisitos do sistema segundo critérios de qualidade

estabelecidos e rastreabilidade. Esta atividade se comunica com o desenvolvimento em momentos estabelecidos no planejamento, que são marcos estabelecidos para aceitar alterações de requisitos.

**Gerenciar Portfólio**: Esta macro atividade fica em um nível superior ao desenvolvimento, sendo que nela as informações são compartilhadas entre os projetos e a alta gerência delibera sobre o andamento e decisões dos mesmos.

Buscando melhorar as atividades relacionadas com a gerência de requisitos, incorporou-se no processo a Gestão do Design, de modo aumentar a qualidade do processo GRE, que se encontra diluído dentro do processo de desenvolvimento GAIA, e também aumentar a qualidade dos requisitos propriamente ditos.