UNIVERSIDADE FEDERAL DO MARANHÃO

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

ESTIMAÇÃO DE MÉTRICAS DE DESENVOLVIMENTO AUXILIADA POR REDES NEURAIS ARTIFICIAIS.

JOSÉ RAIMUNDO DOS SANTOS FONSECA FILHO

São Luís
2003

ESTIMAÇÃO DE MÉTRICAS DE DESENVOLVIMENTO AUXILIADA POR REDES NEURAIS ARTIFICIAIS.

Dissertação de Mestrado submetida à Coordenação
do Programa de pós-graduação em Engenharia de
Eletricidade da UFMA como parte dos requisitos
para obtenção do título de mestre

por

JOSÉ RAIMUNDO DOS SANTOS FONSECA FILHO

Abril
2003

ESTIMAÇÃO DE MÉTRICAS DE DESENVOLVIMENTO AUXILIADA POR REDES NEURAIS ARTIFICIAIS.

José Raimundo dos Santos Fonseca Filho.

Aprovada em ___ de _____ de 2003.

_____

Prof. Dr. Edson Nascimento (orientador)

_____

Prof. Dr. Sofiane Labidi (examinador)

_____

Prof. Dr. Edson Costa de Barros Carvalho Nascimento (examinador)

ESTIMAÇÃO DE MÉTRICAS DE DESENVOLVIMENTO AUXILIADA POR REDES NEURAIS ARTIFICIAIS.

MESTRADO EM ENGENHARIA DE ELETRICIDADE.

ÁREA DE CONCENTRAÇÃO: CIÊNCIA DA COMPUTAÇÃO.

JOSÉ RAIMUNDO DOS SANTOS FONSECA FILHO.

ORIENTADOR: PROF. DR. EDSON NASCIMENTO.

DEDICATÓRIA

Aos meus queridos pais e irmãos.

A minha linda família: Maria, Brena e Ciro.

AGRADECIMENTOS

Ao nosso Senhor Deus Jesus Cristo, por nos mostrar o Caminho.

Aos meus pais, por dedicarem suas vidas à realização pessoal de seus filhos.

À minha esposa, pela carinho dado aos nossos filhos.

Aos professores Edson Nascimento e Maria da Guia, pelo apoio e encorajamento à realização deste trabalho.

RESUMO

Diversas representações do processo de desenvolvimento na Engenharia de *softwares* capazes de, eficientemente, subsidiar a tomada de decisões no gerenciamento de projetos, vêm sendo arduamente pesquisadas. Métricas de *softwares*, modelos de processo e técnicas de estimação têm sido propostos em grande quantidade, tanto devido a características intrínsecas dos *softwares* quanto a características do próprio processo construtivo. Buscando superar algumas das dificuldades de estimação de métricas relacionadas ao processo de desenvolvimento, este trabalho realiza, inicialmente, um estudo de ferramentas voltadas para tal objetivo e que estão disponíveis no mercado. Em seguida, um conjunto de descritores do processo em questão e também um conjunto de atributos básicos dos *softwares* será levantado. A partir de então, é proposto um modelo que represente o processo de desenvolvimento de maneira simples e eficiente. O modelo de processo do desenvolvimento na Engenharia de *softwares* relaciona as características desse processo construtivo a classes de entidades desenvolvedoras, tal que se possa estabelecer um comportamento homogêneo ao processo. Baseado nessa classificação, são relacionados, de maneira direta, métricas (tempo e esforço) de desenvolvimento com os atributos básicos dos *softwares*, definidos por Albrecht, visando a estimação de métricas. O modelo de processo é baseado no mapa de Kohonen e o estimador de métricas será auxiliado por redes neurais *feed forward*. Uma ferramenta de *software* (protótipo) é especificado em Linguagem de modelamento unificada (UML). Esta ferramenta auxiliará a produção de estimativas de tempo e de esforço de desenvolvimento de *softwares*. Comparações de resultados obtidos serão realizadas com os disponibilizados na literatura consultada.

Palavras-chave: estimativa, métrica, rede neural artificial, Análise por pontos de função, COCOMO, molelamento orientado a objetos.

ABSTRACT

Several modeling approaches for the process of development in software engineering able of subsidizing decision making in the management of project are being searched. Metric of softwares, process modeling and estimation techniques have been independently considered either taking into consideration the intrinsic characteristic of softwares or their constructive process. This research proposes a complete, simple and efficient model for representing the whole process of development which, based on a set of features of the process and basic attributes of softwares, yields good estimation metrics (time and effort) of the development of the software still at the beginning of the process. The model relates constructive characteristics of the process to each type of organization, for identifying classes of homogeneous behavior based on Kohonen Neural Network. Directly, from this classification, according to the basic attributes of each software being developed, metrics may be estimated supported by Feedforward Neural Networks. A prototype is specified in Unified Model Language (UML) and implemented to estimate metrics for the development of softwares. Comparisons of the obtained results with those available in literature are presented.

Keywords: metric estimation, neural networks, function point analysis, COCOMO, object model.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

A tarefa de estimação de medidas gerenciais de desenvolvimento na Engenharia de *software* requer o conhecimento sobre o conjunto de componentes constitutivos do *software* a ser construído e do processo a ser executado. Um bom modelo representativo aliado a técnicas eficazes de relacionamento entre tais componentes e suas respectivas medidas gerenciais podem subsidiar eficientemente a tomada de decisões.

1.1 Estimação de medidas gerenciais na engenharia de *softwares*.

Uma forma simples de construção de um artefato tem sido através do método artesanal, em que a manipulação de atributos básicos altera sua estrutura com o objetivo de adequá-lo ao grau solicitado de usabilidade e qualidade. Este método é normalmente denominado de *tentativa-e-erro.* Uma vez que manipulando diretamente o material com o qual o produto será construído os custos normalmente tornam-se proibitivos, o método artesanal é freqüentemente considerado antieconômico.

O aumento nos índices de qualidade do produto, eficiência e produtividade do processo de construção necessita de métodos, ferramentas, organização, ou seja, demanda uma infra-estrutura adequada ao potencial de produção da entidade produtora. Segundo Leite (2000):

> *Quando o desenvolvimento de um artefato incorpora as atividades de análise, design, implementação e avaliação de acordo com princípios, modelos, técnicas, ferramentas e métodos específicos, realizados por uma equipe de especialistas e obedecendo a um planejamento e gerenciamento de custos e prazos, podemos caracterizá-lo como uma engenharia.*

O IIIE (International Institute of Industrial Engineering) atribui como competências da Engenharia de Produção: o projeto, a implantação, a operação, a melhoria e a manutenção de sistemas produtivos integrados de bens e serviços. Compete também especificar, avaliar e estimar resultados, recorrendo-se a várias especialidades do conhecimento como à Matemática, à Física, às Ciências Humanas e Sociais.

APÊNDICES

Fonseca Filho, José Raimundo dos Santos
Estimação de métricas de desenvolvimento auxiliada por redes neurais artificiais/José Raimundo dos Santos Fonseca Filho. _ São Luís, 2003.
86f.:il.
Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Maranhão; 2003.
1. Redes neurais artificiais. I. Título.

CDU 004.032.26

A produção profissional de *softwares* pode explorar a Engenharia em vários aspectos. A Engenharia pode fornecer a base necessária ao estabelecimento de critérios de viabilidade dentro de prazos e orçamentos predefinidos, isto é, subsidiar a administração da produção de *softwares* de modo econômico.

> *Uma abordagem de engenharia ao desenvolvimento de software é caracterizada pela aplicação dos princípios, métodos, padrões e teorias que possibilitam gerenciar, planejar, modelar, projetar, implementar, medir, analisar, manter e aprimorar um sistema de software (Peters e Pedrycz, 1999).*

A Engenharia de Software utiliza conhecimentos de outras duas Engenharias: a Engenharia de Sistemas e a Engenharia de Hardware. Ela tem objetivo predefinido – o estudo do método, das ferramentas e dos procedimentos atinentes à produção de *softwares* (Pressman, 1995).

O método envolve um conjunto de tarefas como planejamento, estimativa de projeto, análise de requisitos de *software* e de sistemas, projeto de dados, arquitetura, algoritmo de processamento, codificação, teste de componentes, integração dos módulos componentes, manutenção e melhoria. Essas tarefas determinam o *modus operandi* da criação, manutenção ou melhoria de *softwares*.

As ferramentas utilizadas na engenharia de *softwares* tentam oferecer aumento na produtividade dos processos. Elas são artefatos de *software* que se portam como facilitadoras da execução das tarefas de engenharia. Apresentam-se em forma de linguagens de programação de alto nível, de utilitários que oferecem informações sobre programas, de simuladores de comportamento de *software,* com depuradores de erros dinâmicos, geradores de código, descompiladores etc. Ferramentas como participam de todas as fases do desenvolvimento, manutenção ou melhoria de *softwares*. Elas podem ser utilizadas no planejamento de sistemas, gerenciamento de projetos, rastreamento de requisitos, medição dos processos de engenharia e seus respectivos produtos e em atividades de apoio (documentação, configuração e categorização).

O procedimento especifica a seqüência de passos a serem seguidos e cada ferramenta a ser utilizada nas atividades de Engenharia de Softwares. Os procedimentos comportam-se como métodos mais abstratos relacionados com a gerenciabilidade do projeto

em execução, constituintes de cada uma das etapas da Engenharia de Software. Independentemente do paradigma adotado para a situação específica de engenharia de *software* existente em uma determinada entidade de *software* (aquela que desenvolve, mantém ou melhora *softwares*), Pressman define três etapas genéricas que compreendem qualquer atividade de engenharia: a **definição**, o **desenvolvimento** e a **manutenção**.

Na **fase de definição** são executados: a **análise de sistemas** - a definição do papel de cada elemento existente no sistema e que será atribuído a um artefato de *software*; o **planejamento de projeto** - em que serão analisados os riscos inerentes, alocados os recursos disponíveis, levantados os custos esperados e estabelecido o cronograma de tarefas a ser seguido; a **análise de requisitos** - detalhamento das funcionalidades que representarão os processos e informações existentes no domínio do problema.

Na **fase de desenvolvimento**, tem-se: o **projeto de *software*** – representação das informações existentes na análise de requisitos em termos do domínio do *software*, como método, estruturas de dados, algoritmos e interfaces; o processo de **codificação do *software*** – conversão para linguagem de programação em computador dos termos estabelecidos no projeto; os **testes** do *software* construído – realização da validação dos requisitos especificados e das funcionalidades solicidadas *versus* construídas na codificação.

A **fase de manutenção** do *software* constitui-se de: **correção** – quando o usuário utiliza o *software*, ele identifica falhas no comportamento requisitado de alguma funcionalidade; **adaptação** – tarefa de modificar os requisitos não funcionais (aqueles que não se referem à atividade fim do sistema) em busca de melhoria de desempenho ou portabilidade; e **melhoramento** da funcionalidade – novas solicitações serão estabelecidas pelo usuário, resultando no aumento dos requisitos funcionais.

1.2 Os processos na engenharia de *softwares*

O processamento realizado na construção de sistemas computacionais pode ser resumido à execução de atividades como administração, orçamento, planejamento, modelagem, análise, especificação, projeto, implementação, testes e manutenção. Tais atividades se utilizarão de uma grande variedade de recursos que possibilitarão a criação e/ou a alteração de seu produto: o *software* (Fenton e Pfleeger, 1997).

Figura 1.1 – Estrutura do processo de engenharia de software (Weber et. alii., 2001).

O processo de *software* é o conjunto de atividades executadas por uma entidade com o objetivo de criar, melhorar ou manter um sistema computacional. Estas atividades podem variar de entidade para entidade, mas podem ser organizadas segundo suas similaridades.

De acordo com esta visão, o processo pode ser Universal, Mundial ou Atômico. Quando o processo tenta representar todos os possíveis projetos ele é chamado Universal. Porém, se ele detalha as especificidades de cada projeto, atentando para suas necessidades, planos e métodos, é dito Mundial. O processo é denominado Atômico quando cada tarefa realizada em um projeto é detalhadamente descrita (Peters e Pedrycz, 1999).

A Figura 1.1 possibilita a visualização dos principais agentes do processo de engenharia de *software*, segundo a norma ISSO/IEC 12207 (Weber et. alii., 2001): processos fundamentais, processos de apoio e processos organizacionais.

Os processos fundamentais envolvem atividades relativas ao estabelecimento do contrato de prestação do serviço (aquisição e fornecimento) e aquelas que compõem o ciclo de vida dos *softwares* (desenvolvimento, operação e manutenção). Os processos de apoio subsidiam a transferência de qualidade ao produto em manipulação, bem como registram, medem, avaliam o desempenho dos processos fundamentais. Processos organizacionais relacionam-se com a administração da entidade desenvolvedora de *softwares*, utilizando os processos de apoio como fonte para o planejamento, direção e controle dos projetos de *software*.

No âmbito dos processos fundamentais, o desenvolvimento ocupa posição de destaque, sendo constituído por atividades como **análise** e **especificação de requisitos**, **projeto**, **codificação**, **integração**, **testes**, **instalação** e **aceitação** relativas à criação de *softwares*.

A **análise e especificação de requisitos de *software*** envolve as atividades de determinar os objetivos de um sistema de *software* e as restrições associadas a ele. Ela deve também estabelecer o relacionamento entre estes objetivos e restrições e a especificação precisa do *software* (Selner, 1999).

A partir dos requisitos técnicos, requisitos de mercado e normas técnicas anteriormente especificados, Selner (1999) também define **projeto de desenvolvimento de *software***.

> *(...) processo responsável pela idealização da solução em termos de: projeto dos programas, projeto de banco de dados, projeto de ferramentas de produção, processos de produção dos programas e processos necessários à implantação do software.*

Segundo Yourdon (1990), a **codificação** é a atividade de construção de módulos dos sistema de *software* através da utilização de uma determinada linguagem; e a **integração** é o ajuste e adequação dos módulos de *software* codificados.

Assim que código do *software* tenha sido gerado, são verificados aspectos lógicos internos e externos relacionados com a funcionalidade requisitada. Esta atividade é descrita por Pressman (1995) como a fase de **testes** do processo de desenvolvimento de *software*.

A **instalação** e **configuração** são as atividades que visam implantar o *software* no computador para que ele possa ser utilizado pelos usuários. A instalação requer que o *software* seja armazenado no lugar correto e a configuração, que os *drivers* de dispositivos de *hardware* sejam configurados adequadamente de tal modo que o *software* funcione corretamente. Esta atividade pode ser realizada tanto por usuários como pelos engenheiros ou programadores do *software* (Leite, 2000).

A **aceitação** ou utilização é a atividade final do desenvolvimento de *softwares*. Uma vez que este é construído para auxiliar pessoas na realização de suas tarefas, sua utilização deve seguir o modelo de interação especificado durante o seu projeto que deve estar descrito no manual de usuário (Leite, 2000).

1.3 O *software*

Numa abordagem de engenharia, o processo economicamente gerenciado objetiva a construção eficaz e eficiente de um produto, que no caso da Engenharia de Software é o sistema computacional, ou seja: o *software*. Neste contexto, Diverio e Menezes (1999) definem um *software* como:

> *(...) conjunto estruturado de instruções que capacitam uma máquina a aplicar sucessivamente certas operações básicas e testes sobre os dados iniciais fornecidos, com o objetivo de transformar estes dados numa forma desejável.*

A máquina citada pode ser entendida como um sistema organizado de componentes físicos que se destinam a realizar um conjunto pré-estabelecido de operações básicas identificadas por comandos relativos à uma determinada linguagem de comunicação com o meio externo. A máquina e a linguagem existem em qualquer sistema computacional. Aliados a esses, outros recursos como: plataforma, metodologia, pessoas etc., são também utilizados pelo processo de engenharia de *software*.

Pressman (1995) estabelece uma classificação de *softwares* segundo a área de aplicação. Ele os divide em ***softwares* básicos** – desenvolvidos para dar suporte a outros *softwares*; ***softwares* de tempo real** – intervém no mundo real ao mesmo tempo em que os eventos reais ocorrem; ***softwares* comerciais** – processam informações administrativas (geralmente utilizam grandes bancos de dados); ***softwares* científicos e de engenharia** –

utilizam grandes quantidades de algoritmos e são, por vezes, usados para realização de simulações e outras aplicações interativas; ***softwares* embutidos** – geralmente de funcionalidade limitada, residentes na memória principal do computador (usados em sistemas de controle industriais e de consumo); ***softwares* de computador pessoal** – desenvolvidos para propósitos gerais para execução em computadores pessoais; e ***software* de inteligência artificial** – utilizados para reconhecimento de padrões, jogos, demonstrações de teoremas e outras atividades relacionadas com o conhecimento.

A partir de uma visão funcional, o *software* pode ser abstraído como um conjunto de atributos básicos que compõem uma estrutura lógica visando oferecer um comportamento previamente solicitado e esperado. Porém, o processamento a que ele é submetido, determina a sua eficácia, isto é, sua qualidade.

1.4 Estimativas no desenvolvimento

A realização de estimativas em relação ao desenvolvimento de *softwares* é uma importante atividade dos processos de apoio na Engenharia de Software*,* Fig. 1.1. Seus prognósticos possibilitam tecer-se projeções a respeito de projetos que serão executados por uma entidade desenvolvedora, servindo como base para a tomada de decisões. Porém, caso a estimativa utilizada seja ineficiente, o dano causado à entidade poderia ser maior que a não existência de estimativas.

A produção de estimativas para a tomada de decisões gerenciais necessita de medidas capazes de fornecer uma noção completa tanto do sistema em construção quanto de seu respectivo processo produtivo.

A medida é fator preponderante no estudo das várias tarefas componentes da engenharia. Ela subsidia as aferições dos processos e dos atributos do produto. Uma medida pode ser vista como um parâmetro que possibilita o mapeamento de um conjunto para outro, onde o segundo conjunto forneceria melhores informações a respeito do sistema modelado que o primeiro. Ela possibilita comparações, julgamentos e decisões gerenciais e é elemento básico na execução de estimativas de projeto de desenvolvimento ou de manutenção de *softwares*.

No contexto da Engenharia de Software, as medidas são utilizadas para quantificar atributos específicos do produto do processo de desenvolvimento, relacionando-se com:

recursos humanos - quantidade de profissionais que serão utilizados na execução do projeto; *cronograma* - tempo em que cada profissional será necessário; *escalonamento do pessoal* - quando esses profissionais serão necessários; e *orçamento* - quanto custará o projeto.

Estimar medidas do processo de desenvolvimento de *softwares* é uma tarefa que apresenta dificuldades bem variadas. Primeiramente, o produto deste processo tem natureza intangível com atributos pouco definidos. O *software* pode ser compreendido de acordo com a criatividade e experiência humanas onde muitos níveis de abstração podem ser utilizados. Assim, não sendo claro e objetivo o modelo de *software* utilizado, as diferenciadas expectativas a respeito do *software* podem não satisfazer uniformemente àqueles envolvidos em sua construção.

Por outro lado, a flexibilidade do processo de desenvolvimento possibilitam diferenciados níveis de padronização do gerenciamento produtivo nas entidades desenvolvedoras de *software*. Desta forma, podem ser produzidos modelamentos distantes do sistema real, e, conseqüentemente, as técnicas estimativas atualmente aplicadas e baseadas nesses modelamentos ofereceriam resultados ineficientes, podendo chegar até mesmo a serem inadequados. Deve-se então reunir um conjunto de características que, de maneira geral, capture as especificidades dos processos de desenvolvimento.

Em geral, o registro de eventos ocorridos durante o desenvolvimento é subjetivo, às vezes inexistente, ou pior, falacioso. As medidas utilizadas nos processos em que existe um mínimo gerenciamento disponibilizam pouca clareza a respeito da estrutura e comportamento do desenvolvimento. Logo, além de ser objetiva, a métrica de *software* deve ser simples, senão não será bem usada.

1.5 Objetivos e metodologia do trabalho

Devido à complexidade do *software* e de seu respectivo processo produtivo, muitos modelos representativos da engenharia de *softwares* foram criados. O objetivo perseguido é subsidiar a tomada de alguma decisão pertinente à gerenciabilidade do projeto construtivo, evolutivo ou de manutenção de *softwares.* Porém, vários obstáculos impostos a tal tarefa ainda precisam ser superados.

Esta dissertação propõe definir um modelo que represente o processo de desenvolvimento na Engenharia de Software e seu produto, fundamentado em requisitos

levantados na literatura e em algumas das características identificadas e selecionadas em técnicas largamente utilizadas na atualidade, como a Análise por Pontos de Função (FPA) e o Modelo de Custo Construtivo (COCOMOII).

São objetivos desta dissertação:

- levantar um conjunto de descritores de desempenho no processo de desenvolvimento realizado por entidades produtivas de *software* (instituição responsável pelo desenvolvimento de um *software*);
- identificar os atributos básicos dos *software* capazes de representá-lo a partir de suas funcionalidades;
- classificar as diferentes entidades produtivas de *softwares* de acordo com os descritores de desempenho levantados, para que se possa fornecer estimativas a partir de uma base menor de dados empíricos;
- investigar a utilização de redes neurais no relacionamento de características essenciais do *software* (atributos básicos) e medidas de seus respectivos processos construtivos (tempo e esforço de desenvolvimento);
- construir uma ferramenta computacional, baseada no modelamento em proposição, que permita subsidiar a tomada de decisões no gerenciamento do desenvolvimento de *softwares*, de forma padronizada e eficiente;
- oferecer estimativas de esforço e de tempo de desenvolvimento nas fases inicias do ciclo de vida dos *softwares.*

Para a obtenção do modelo de mencionado, esta dissertação dividirá o problema de estimação de métricas de *software* em duas partes relativas às seguintes visões:

- **visão a partir dos componentes do processo** - em que serão considerados os aspectos relativos às atividades e atributos componentes do processo de desenvolvimento;
- e **visão a partir do produto em desenvolvimento** - em que serão considerados apenas os fatores relacionados ao produto *software* como determinantes das alterações nas variáveis de desempenho do processo de desenvolvimento.

A partir da visão dos componentes do processo, o problema de estimação de métricas de desenvolvimento se preocupará com o fornecimento destas medidas em processos diferenciados, constituídos de fatores que caracterizarão entidades heterogêneas. Ou seja, será admitido que variações qualitativas ou quantitativas nos fatores determinantes do desenvolvimento desempenhado pela entidade produtora de *software* gerarão variações quantitativas no processo de desenvolvimento de um mesmo produto.

Os fatores relativos ao processo serão definidos a partir dos requisitos levantados a partir das visões de Yourdon (1990), Machado(1990), Pressman (1995) e Büren e Koll (1999). Espera-se que isto leve à idealização de um modelo capaz de fornecer uma visão ampla do processo de construção de *softwares*.

A partir da visão do produto, admitir-se-á que uma determinada entidade se utilize de um grupo de recursos cujo comportamento seja pouco variável durante a desenvolvimento de um sistema computacional. Neste *ambiente* específico de produção, os *softwares* projetados e implementados estariam submetidos a condições produtivas aqui definidas como *condições homogêneas.* Isto é, as únicas diferenças entre o desenvolvimento de um *software* e de outro seriam determinadas por suas próprias características constitutivas ou atributos básicos, não levando em conta variações tecnológicas em seu processamento.

Os atributos básicos relativos ao produto serão os definidos pelo modelo funcional de Albrecht (IFPUG, 1994), que define o *software* como um conjunto de funções agrupadas de forma predefinida para o fornecimento da funcionalidade requisitada pelo usuário. Após sua definição e análise, serão utilizadas as características do modelo consideradas importantes pela literatura para propor um modelo que buscará relacioná-las com o tempo e com o esforço de desenvolvimento.

Na implementação de tal modelo, serão utilizadas redes neurais artificiais treinadas para proceder os seguintes papéis:

- na visão do processo, essas redes serão responsáveis por classificar as diferentes entidades desenvolvedoras de acordo com as características de seu processo e ambiente produtivo;
- a partir da visão do produto, as redes serão utilizadas para a determinação dos relacionamentos existentes entre os aspectos básicos do *software* e suas métricas produtivas.

Considerando o modelo especificado, será construída uma ferramenta computacional que permitirá a realização de estimativas de métricas de desenvolvimento de *softwares* baseados em diferentes constituições de equipes, submetidas a variadas formas organizacionais, utilizando-se de diversos ambientes de produção. Os resultados obtidos serão comparados com alguns outros fornecidos pela literatura.

1.6 Apresentação da dissertação

O Capítulo 2 apresenta maiores detalhes a respeito da medida como base para a engenharia de *softwares,* onde é descrita uma classificação de métricas de *software* com alguns exemplos. Também é tratada o modelamento na Engenharia de *software* e métodos para estimação caracterizados por Wu (1997) e Chulani et. alii. (1998). Os requisitos para a estimação de medidas gerenciais na engenharia de *softwares* são apresentados a partir das visões de Yourdon (1990), Machado(1990), Pressman (1995) e Büren e Koll (1999). Em seguida, mostra os modelos atualmente utilizados para a estimação de medidas de *software*. Especificamente, a visão a partir do processo apresenta o modelo COCOMO II (Boehm et. alii., 1998), e, a visão a partir do produto mostra o modelo de Albrecht (IFPUG, 1994).

O Capítulo 3 descreve a proposta de modelamento de Engenharia de Software em que são discriminados os principais fatores contidos nos modelos mencionados. Nesse capítulo, é considerada a visão de Chiavenatto (2001) a respeito das organizações administrativas, definidas pela Teoria Geral da Administração. Utilizando-se a sistematização do processo de desenvolvimento proposta por Büren e Koll (1999), é iniciada a construção do modelamento objetivado neste trabalho.

No Capítulo 4, está especificado um estimador de tempo e esforço de desenvolvimento de *softwares*, cuja arquitetura de classes seguirá desenhada. A implementação do especialista estimador, contido neste sistema, está representada pelo protótipo construído em linguagem Java. Testes e comparações fornecem um juízo de valor a respeito do protótipo desenvolvido.

Finalmente, no Capítulo 5 expõem-se as conclusões desta dissertação e suas projeções para o futuro. As Redes Neurais Artificiais (estrutura, comportamento, formas de ajuste de estrutura: treinamento e aprendizado) estão abordadas no Apêndice A. Duas arquiteturas largamente utilizadas são vistas: o mapa auto-organizado de Kohonen e a arquitetura *feed forward* com aprendizado *back propagation*.

CAPÍTULO 2

MEDIDA, ESTIMAÇÃO E MODELAGEM NA ENGENHARIA DE *SOFTWARES*.

A Engenharia de Software é representada pela *aplicação de uma sistemática, disciplinada e quantificada abordagem para o desenvolvimento, a operação e a manutenção de software* (Zuze, 1998). É a *(...) disciplina envolvida com a produção e manutenção sistemática de software que são desenvolvidos com custos e prazos estimados* (Leite, 2000). Destas definições, destacam-se a necessidade de quantificação dos processos estabelecidos pela engenharia de *softwares*, bem como a estimação como base para a tomada de decisões.

2.1 A medida como base para a engenharia de *softwares*.

Metrologia, a partir de sua origem grega (*metron*: medida + *logos*: ciência) é o termo utilizado para designar a ciência da medição. Segundo o manual de metrologia produzido pela CNI/COMPI (2000), medição é o *conjunto de operações que tem por objetivo atribuir um valor (número) a um determinado fenômeno*. Tal valor, medida ou métrica tem o objetivo de fornecer uma complementação ao sistema sensorial humano tal que esse possa registrar dados e informações relacionados ao fenômeno em análise.

Métricas são escalas de unidades padronizadas que servem para apurar a quantidade que essas unidades participam na composição de uma grandeza. O estabelecimento de uma unidade de medida (grandeza cujo valor é definido como exatamente um) possibilita a comparação entre duas medições da mesma grandeza, facilitando a análise econômica e qualitativa dos aspectos produtivos que as envolvem. Na medição são estabelecidas unidades de medidas padrão (uma referência com a qual devem ser comparados todos os outros exemplos de métricas da grandeza) buscando a fácil e uniforme utilização por diferentes analistas.

Como em qualquer outra área da ciência, na Engenharia de Software também se necessita de um conjunto de métricas para a quantificação do produto e de seu respectivo processo de manufatura, pois seu principal objetivo é o estabelecimento de princípios que possibilitem o processamento econômico, confiável e eficiente (desenvolvimento ou

manutenção) de programas, baseando-se em técnicas que garantam seus aspectos de qualidade e produtividade (Pressman, 1995).

A métrica de *software* é ferramenta básica no acompanhamento, planejamento e controle das atividades constituintes dos processos de engenharia de *softwares*. Como menciona Desharnais e Abran (1999), *medidas confiáveis conduzem a uma melhor compreensão dos fatores de produtividade e suportarão o processo decisório*. Quando bem definidas, as métricas são úteis para a determinação de estimativas do tamanho final do *software*. Outros fatores relativos às atividades produtivas do processo de desenvolvimento de sistemas computacionais, como o tempo e o esforço despendido, a produtividade da equipe desenvolvedora, o grau de influência de ferramentas utilizadas na implementação e a qualidade final do *software* desenvolvido, podem ser determinados a partir dessas medidas.

Mas o *software* e sua produção têm características muito próprias de intangibilidade, oriunda de fatores como complexidade e subjetividade (Jones, 1991). Há incerteza quanto ao seu tamanho final até que ele seja completamente escrito. Chulani et. alii. (1998) afirmam que quanto mais ele cresce em tamanho e importância mais aumenta em complexidade, tornando-se mais difícil realizar-se a previsão de seu custo, esforço, produtividade, tempo de desenvolvimento etc. Desta forma, uma grande quantidade de métricas tem sido proposta e utilizada no mercado, o que torna o gerenciamento do processo de produção uma questão ainda mais problemática.

2.1.1 Classificação de métricas de *software*

As métricas de *software* podem ser classificadas em: **diretas** e **indiretas**. Métricas diretas estão relacionadas com as características físicas do sistema computacional ou com seu processo. São facilmente utilizadas, mas são dependentes da tecnologia, necessitando que os projetos sejam consideravelmente similares ao projeto estimado. Isto é, para especular os valores esperados correspondentes ao desenvolvimento de novas funcionalidades ou à manutenção de um determinado *software*, elas utilizam informações relativas à constituição do produto, fundamentadas em registros históricos ou em avaliações advindas da experiência de especialistas.

São métricas diretas: milhares de linhas de código (KLOC) implementadas, quantidade de defeitos identificados, tamanho de memória utilizada, tempo de alocação de CPU, tempo de desenvolvimento, quantidade de pessoas alocadas durante o desenvolvimento,

custo e esforço totais de produção, número de instâncias de classes utilizadas, quantidade de classes-filhas etc.

As métricas indiretas são medidas que tentam capturar aspectos mais subjetivos dos componentes da engenharia de *softwares*, como a complexidade de produção a partir da ótica do desenvolvedor, ou a funcionalidade do programa pela ótica do usuário. São especialmente úteis para a análise de produtividade e para a construção de modelos de estimação (Olligny et. alii,1999). São independentes da tecnologia empregada e se baseiam em indicadores de aspectos relativos à complexidade do programa, percebida pelos agentes participantes do processo de desenvolvimento (Pressman, 1995; Goodbrand, 1997). Porém, o seu estabelecimento é considerado uma atividade de alto grau de dificuldade. Wu (1997) atribui esta dificuldade à natureza do programa: intangível, invisível e intratável. Relacionam-se também com aspectos como funcionalidade, complexidade e confiabilidade.

Dentre as medidas indiretas existentes, pode-se citar: número de pontos de função (IFPUG, UKSMA, COSMIC etc.); complexidade ciclomática de McCabe; e medidas de Halstead. São medidas que tentam capturar a qualidade, a funcionalidade, a complexidade, a eficiência, a confiabilidade, a manutenibilidade do produto e seu processo.

A dificuldade de determinação de métricas confiáveis e viáveis está na medição dos fatores componentes dos seus processos constitutivos do desenvolvimento e dos fatores representativos do produto de *software*. Estes fatores são muito peculiares a cada projeto, produto e processamento. Tal dificuldade alimenta a produção de uma grande quantidade de trabalhos nesta área, cada um com características próprias e diferentes performances. As diferentes maneiras de modelar o processo e o produto na Engenharia de Software podem ser estudadas em trabalhos como Boehm et. alii. (1998); Goodbrand (1997); Lokan e Abran (1999); Abran (1994); Abran e Robillard (1996); dentre muitos outros.

Outra classificação das métricas de *software* as dividem em primárias e derivadas. As métricas primárias estão relacionadas a características obtidas diretamente do produto ou processo de desenvolvimento. As métricas derivadas são medidas definidas por relações criadas a partir das métricas primárias. São exemplos de métricas primárias: KLOC, número de pontos de função, complexidade ciclomática, tempo de desenvolvimento etc. Medidas derivadas têm como exemplo: esforço de desenvolvimento, produtividade da equipe, qualidade do processo, taxa de produção de *software*, dentre outras.

A importância de uma medida de *software* é dada pela necessidade de sua medição em relação aos objetivos da entidade desenvolvedora. Os principais objetivos da utilização de métricas de *software* são: a estimação do custo de produção de projetos de desenvolvimento; a estimação do tempo necessário à execução do mesmo projeto; e a medição da qualidade do produto resultante do desenvolvimento.

A seguir, serão consideradas algumas das métricas mais freqüentemente citadas na literatura, mais especificamente: o número de linhas de código e o número de pontos de função, ambas referidas como medidas relacionadas ao tamanho do *software*.

2.1.2 Algumas métricas de tamanho de *software*.

A métrica de tamanho de *software* mais utilizada é o número de linhas (LOC) ou milhares de linhas (KLOC) de código fonte. A utilização de medidas baseadas em LOC se baseia na hipótese de que *softwares* com funcionalidades similares fornecerão tamanhos em LOC também similares. Trata-se de uma métrica direta e primariamente obtida.

No entanto, antes de se proceder à contagem do número de linhas de um *software*, dever-se-á definir claramente o que se poderá considerar como linha de código, pois elas pode conter comandos, laços, comentários, e outras seqüências de caracteres que não necessariamente deveriam ser consideradas.

Dentre as várias padronizações existentes, o *software* Engineering Institute (2000) fornece uma estrutura que exemplifica como se deve proceder à contagem de LOC, abaixo enumerada:

- identificação dos principais atributos que definem o objeto de medição;
- identificar os valores de cada atributo que diferentes usuários da medida vão querer incluir/excluir de suas medições, garantindo que esses valores seja mutuamente exclusivos;
- realizar uma averiguação dos principais atributos e seus respectivos valores, tal que os valores inclusos/excluídos na medição possam ser explicitamente identificados;
- garantir que se possa compreender a importância da cobertura e visibilidade de cada atributo para todos os usuários;
- identificar os valores que serão incluídos juntos com os valores que serão excluídos, em cada atributo;

- identificar e registrar todos as regras adicionais e as exceções seguidas no processo de medição (definição);
- identificar e registrar os valores que serão colecionados e guardados para cada medição individual (especificação);
- realizar a medição conforme a definição e especificação;
- agregar os resultados à medição e gerar relatórios;
- anexar a definição e a especificação para cada conjunto medição-relatório.

Devido à característica "linha de código" existir em todos os *softwares*, o número de linhas de código é facilmente obtido, simples de utilizar e conceituar, tanto por parte do usuário quanto do cliente. Porém elas são altamente suscetíveis à variação da linguagem de programação, não capturam a complexidade do produto, necessitam de uma grande quantidade de características do processo e do produto para produzir estimativas eficientemente.

Outra métrica de tamanho do *software* largamente utilizada é o número de pontos de função, que tenta relacionar o número de funcionalidades requisitadas pelo usuário ao "tamanho funcional" do *software* desenvolvido. Ela objetiva capturar o grau de funcionalidade oferecido e entregue ao usuário. Seu processo de medição é realizado na fase inicial do projeto.

O uso do modelo de funcionalidade na visão do usuário é o responsável por tornar FPA uma métrica independente da tecnologia aplicada ao processo de desenvolvimento ou de manutenção. Esta característica a torna um atraente estimador do tamanho de programas, pois pode ser usada nas fases iniciais do ciclo de vida do desenvolvimento, além de ser considerada de simples compreensão por usuários não-especializados (Wu, 1997).

Os pontos de função ajustados são usados com vários propósitos (Lokan, 1999):

- como fator de normalização, facilitando a análise de diferentes projetos por diferentes especialistas de maneira uniforme;
- como variável dependente em modelos que objetivam seu relacionamento com o esforço de desenvolvimento;
- como unidade de conversão entre diferentes métricas de programa.

Entretanto, o método FPA é adequado a somente parte do domínio dos programas existentes, os conhecidos como Sistemas de Gerenciamento de Informações (SGI), pois devido a sua grande granulosidade, subestima projetos no domínio do tempo-real (St-Pierre et. alii, 1997; Oligny et. alii, 1999).

Tentando estabelecer um mapeamento entre complexidade e tamanho total das funcionalidades entregues ao usuário, fundamentando-se no esforço de desenvolvimento envolvido, o modelo em que se baseia FPA é composto por várias escalas de transformação implícitas, algumas matematicamente inválidas (Abran e Robillard, 1996).

A uniformidade objetivada por esse método é comprometida pela utilização de fatores subjetivos *CGS* (Características Gerais dos Sistemas) que possibilitam inúmeras interpretações e dificultam a obtenção de resultados consistentes (VanDoren, 2001).

Lokan (1999) chega a sugerir que da maneira que está definido, as *CGS* não deveriam ser usadas para o ajuste de complexidade, pois não consegue incrementar a qualidade da métrica em relação ao esforço e produtividade envolvidos. Em seu estudo, afirma ainda que talvez somente algumas características fossem relevantes e que algumas delas são mais importantes que outras (para diferentes projetos) e que elas pouco variam.

Além de outros tantos fatores, as desvantagens mencionadas propiciaram o aparecimento de diversos refinamentos e variações de FPA (MKII, Feature points, FFP etc.), fato que levou a ISO (International Standards Organization) a promover um estudo, iniciado em 1993, para o desenvolvimento de um padrão de medição funcional, chamado Functional Size Measurement (FSM) sem considerar ajustes de complexidade (VanDoren, 2001).

2.2 Estimação e modelamento na Engenharia de Software.

Após escolher o conjunto de medidas de *software* mais adequado a suas características, a entidade desenvolvedora deveria criar uma base de dados históricos contendo variadas combinações de produto e processo caracterizadas por tais medidas, e, utilizando-se de algum método para representação da engenharia de *softwares*, poderia realizar estimativas gerenciais a respeito do projeto a ser executado.

2.2.1 Métodos para estimação na Engenharia de Software.

Vários são os métodos propostos para a estimação de medidas relativas ao processo de desenvolvimento. Eles podem se fundamentar, segundo Wu (1997), nas diferentes abordagens que se seguem.

A primeira abordagem é fundada no **julgamento fornecido por especialistas** capazes de prognosticar eventos futuros, baseando-se na experiência vivida em semelhantes projetos. Este método possui as seguintes vantagens: são sensíveis às diferenças entre os projetos passados e futuros; permitem avaliar o impacto da utilização de novas tecnologias no novo projeto. Porém, são altamente subjetivos e implícitos, além de não serem quantificáveis e documentáveis.

Alternativamente, a estimação também pode ser realizada através de **analogia**, onde a experiência mencionada acima é mais formalmente registrada. Ela depende da qualidade e da quantidade de informações registradas. Suas vantagens são as mesmas da estimação por julgamento especialista, mas num grau pouco mais rígido. É intuitivo e menos subjetivo, mas depende de bons descritores e de certo grau de similaridade entre os projetos.

Outra possibilidade é obtida através de abordagem ***top-down***, em que os aspectos gerais dos projetos são considerados. O projeto é dividido em partes menores chamadas componentes para aumentar a gerência. É orientado às atividades do nível de sistema e, ainda, fácil e rápido de implementar. De outro lado, subestima a dificuldade dos problemas em baixo nível e apresenta pouca possibilidade de detalhamento.

Inversamente à abordagem anterior, o método de obtenção de estimativas baseadas na visão ***bottom-up*** utiliza o conhecimento em torno dos componentes do projeto para estima-lo por inteiro. É mais flexível, pois permite a manipulação das partes componentes do projeto. É também mais estável, pois o erro é distribuído entre essas partes componentes. De difícil integração e gerenciamento, não permite sua utilização nas fases iniciais do ciclo de desenvolvimento, sendo mais resistente à execução.

O método de estimativa pode ainda se originar de uma **visão algorítmica**, que pressupõe a construção de equações matemáticas oriundas de observações empíricas. Permite um alto grau de repetibilidade, é fácil de modificar, eficiente e objetivo, porém, é avesso a fatores fora do padrão, pois uma imprecisão na entrada gera inconsistências na saída.

Para Wu (1997) e Johnson (1998) a utilização de métricas que levem em conta diferentes métodos de estimação é preferida, pois possibilita a soma das vantagens pertencentes a cada um deles, capturando uma ampla quantidade de visões da realidade de desenvolvimento para a composição de um modelo consistente.

No estudo de Chulani et. alii. (1998), é mostrado que as técnicas e modelos de estimação são utilizadas para predizer o custo, análise de risco, controle e planejamento de projetos e análise de investimento em melhoria de *software*. Em seu trabalho, Chulani et. alii. divide as técnicas de estimação da seguinte forma: as baseadas em modelamento, as relacionadas à avaliação especialista, aquelas que utilizam técnicas de aprendizagem artificiais, as baseadas em regressão e as que usam a composição *bayesiana*.

A estimação baseada em modelamento é realizada através do mapeamento de características do processo de desenvolvimento ou de seu produto para um modelo, geralmente proprietário, que representa uma visão da Engenharia de Software. Exemplos de técnicas deste grupo são, dentre outras:

- **SLIM** – desenvolvida por Putman, é utiliza a curva de Rayleigh para prever o esforço de desenvolvimento em função do tempo de desenvolvimento decorrido;
- ***checkpoint*** – criada por Jones em 1997, é uma técnica que realiza estimativas em nível de projeto, de fase, de atividade e de tarefa, baseando-se no conhecimento adquirido através dos projetos previamente executados;
- **PRICE-S** – criado para uso interno na RCA, consiste de três sub-modelos: *aquisição*, em que o o custo e o cronograma de desenvolvimento são estimados; *tamanho*, que prediz o tamanho final do *software*; e *custo*, que realiza estimativas de custo durante as fases de iniciais do ciclo de vida do desenvolvimento;
- **COCOMO II** – técnica de estimação largamente utilizada, é baseada em modelo construtivo (com o uso de regressão estatística).

As técnicas baseadas na avaliação especialista fazem uso de informações não quantificáveis, obtidas empiricamente. É altamente susceptível ao contexto, pois o aumento de informações utilizadas não garante que a avaliação seja melhorada. Neste grupo estão técnicas como: **DELPHI –** criada pela The Rand Corporation no final dos anos quarenta, que utiliza o consenso como meta para fornecimento de uma avaliação; e **WBS** (Work Break down Structure) - esta técnica organiza hierarquicamente o produto de desenvolvimento tal que se

possa colecionar as informações relativas a cada componente, facilitando a realização de estimativas.

Técnicas para estimação baseadas no aprendizado artificial utilizam redes neurais artificiais, conjuntos nebulosos, algoritmos genéticos, e outras formas de inteligência artificial para registrar o conhecimento relativo aos projetos previamente executados. Estas técnicas possuem características interessantes como: reconhecimento de padrões, estimação de funções não-lineares, tolerância a falhas (Shaw e Simões, 1999).

As técnicas baseadas na dinâmica admitem que as características do processo de desenvolvimento de *software* variam durante seu ciclo de vida. Isto é, que os requisitos do usuário, o custo do projeto, a experiência da equipe, os recursos tecnológicos, variam ao longo do desenvolvimento dos sistemas. Neste grupo está a **Abordagem dinâmica de sistemas** - desenvolvida por Forrester (1961), que representa seus componentes por nós interconectados por linhas ou fluxos realimentados. É matematicamente modelado por equações diferenciais de primeira ordem.

A regressão é uma das técnicas mais freqüentemente utilizadas na realização de estimativas de métricas de *software*. Existem dois modelamentos possíveis: o **convencional**, em que é usado o método dos mínimos quadrados aplicados a uma relação linear composta de coeficientes relacionados com o fenômeno estimado; e o **robusto**, que utiliza o modelo dos mínimos quadrados médios para a redução de problemas existentes no modelo convencional como a impossibilidade de estimação com falta de informações. Utilizam a regressão modelos como COCOMO II, SLIM, *Checkpoint*.

A composição *bayesiana* é uma abordagem que tenta realizar combinações entre diversas técnicas para a formulação de um modelo mais apropriado de estimação de métricas. Baseando-se no teorema de Bayes, ele produz estimativas que consideram estados anteriores aliados às informações de controle como variáveis de entrada. Neste grupo estão técnicas como COQUALMO e COCOMO II.1998.

2.2.2 Requisitos para a estimação de medidas gerenciais na Engenharia de Software.

Para o atendimento da validade e precisão que devem estar presentes em todo tipo de modelo, deve-se orientar-se por alguns preceitos ou requisitos. A proposição de um modelo

que subsidie a realização de estimativas de métricas de *software* em projetos que ainda serão executados não é uma exceção.

Há algum tempo tem-se estudado maneiras de produzir modelos representativos da engenharia de *softwares*. Yourdon (1990) propõe quatro diretrizes a serem seguidas na criação de boas estimativas relativas ao desenvolvimento de programas:

- as unidades de estimativas devem ser tão pequenas quanto possível: isto permite que a precisão das medidas utilizadas seja maior. Entretanto, deve-se atentar para o custo em obtê-las;
- as unidades de trabalho devem apresentar um alto grau de independência: unidades interdependentes dificultam a criação de modelos de processo;
- deve-se considerar o fator de comunicação: o que incrementa o custo de produção da medida; e
- deve-se distinguir trabalho novo de trabalho aproveitado: visa adequar o modelo estimador à experiência obtida em projetos anteriormente executados.

Segundo Machado (1990), a produtividade de desenvolvimento de *software* é afetada pelos seguintes fatores, mais analiticamente dispostos:

a. Linguagem de codificação: as linguagens possuem diferentes potenciais de expressão, baseadas em sua granulosidade, significância e estrutura;

b. Tamanho do *software*: um *software* de grande magnitude possui uma quantidade de atividades periféricas ou administrativas também grande, como: documentação, integração, planejamento, testes para remoção de erros;

c. Experiência da equipe: expressa o grau de habilidade da equipe com a linguagem utilizada, com o ambiente de desenvolvimento, com os métodos e técnicas de análise, com as áreas de negociação;

d. Metodologia: a padronização do método de desenvolvimento de *softwares* possibilita a automatização de atividades componentes do processo produtivo e certa uniformização das soluções. Dada sua sistematização, combate a complexidade no contexto das soluções possíveis oferecidas ao problema;

e. Ambiente de desenvolvimento e Ferramentas CASE: executa atividades freqüentemente necessárias, aumentando a rapidez de produção, facilitando a revisão e uniformizando os procedimentos;

f. Expansão/Manutenção: é um aspecto produtivo que diferencia todos os outros relativos ao desenvolvimento de *software*;

g. Reutilização de código: transforma o processo de desenvolvimento de *software* em uma atividade de integração de partes já implementadas;

h. Circunstâncias geográficas: impacta sobre o grau de comunicação e gerenciabilidade da equipe;

i. Remoção de erros: permite a identificação de erros de implementação do *software*;

j. Recursos de codificação: facilidades fornecidas pelo ambiente de desenvolvimento;

k. Organização da equipe: forma de estruturação da equipe, isto é, aspectos hierárquicos, matriciais etc.

l. Satisfação da equipe: tenta medir o moral da equipe, isto é, a motivação, forma de recompensa, gosto pelo serviço, incompatibilidades interpessoais, ganhos sociais etc.

Várias abordagens foram utilizadas na tentativa de se realizar estimativas que pudessem seguir diretrizes e requisitos como os mencionados anteriormente. Notadamente, segundo Yourdon (1990), muitas das estimativas utilizadas comercialmente se relacionam ao desempenho de pessoas executando tarefas pertinentes ao desenvolvimento de programas, utilizando-se de componentes de trabalho muito heterogêneos, o que se lhe impõe uma árdua dificuldade.

Pressman (1995) enumera uma série de requisitos necessários para a realização de estimativas confiáveis. Segundo ele, deve-se atrasar ao máximo a realização de estimativas, pois, com mais subsídios, erra-se menos. O modelo utilizado deve ser o mais simples possível, devido à redução do esforço extra na realização da atividade de medição. Além disso, o modelo utilizado deve ser empírico e automático.

De forma concisa, Pressman (1995) afirma que a estimação de esforço requer experiência, boas informações históricas e comprometimento com o processo de medição implantado. Para ele, deve-se considerar os seguintes fatores, como caracterizadores do processo de construção de *softwares*:

a. Complexidade do Projeto: empiricamente, possui uma forte relação com o esforço de desenvolvimento, mas deve ser medido relativamente à experiência da equipe com projetos específicos passados;

b. Tamanho do Projeto: mais que a escalabilidade fornecida pela quantidade de componentes do *software*, os efeitos da inter-relação entre maiores quantidades desses componentes gera um aumento não linear no esforço;

c. Estrutura do Projeto: a organização dos componentes do projeto facilita a utilização das funções que comporão a aplicação resultante.

Para Büren e Koll (1999) o processo de estimação deveria respeitar os seguintes requisitos:

- completude – incluir todas as atividades componentes do processo de desenvolvimento além de todos os produtos criados por eles;
- flexibilidade – adaptar-se a qualquer tipo de projeto;
- detalhamento – fornecer estimativas para todas as atividades componentes do processo;
- continuidade – abarcar todo o ciclo de vida do processo;
- experiência – ser fruto do conhecimento adquirido pelos especialistas durante sua interação com os processos previamente executados;
- rastreabilidade – poder ser acompanhada em cada uma de suas atividades;
- reprodutibilidade – fornecer resultados próximos, quando estimativas forem realizadas nas mesmas condições, para mesmos produto e ambiente, e por pessoas diferentes;
- avaliabilidade - permitir sua comparação com os valores reais do processo de desenvolvimento.

Muitos modelos foram criados para representar a engenharia de *softwares*. A partir de diferentes visões, utilizando-se variadas formas para fornecer subsídios à estimação de variáveis do processo de desenvolvimento, muitos modelos atualmente utilizados tentam atingir os requisitos aqui mencionados.

No próximo capítulo, serão tratados dois dos mais utilizados modelos de engenharia de *softwares*. O modelo criado por Boehm (BOEHM et. alii, 1998) utiliza um considerável número de variáveis do processo de desenvolvimento, levando em conta fatores relativos às tecnologias empregadas. O modelo criado por Albrecht (IFPUG, 1994) utiliza um conjunto de variáveis que tentam capturar a funcionalidade do *software*, a partir da visão do usuário.

2.3 Modelos de engenharia de *software* mais utilizados atualmente.

O processo de transformação é o "uso de recursos para mudar o estado ou condição de algo para produzir outputs" (Slack et. alii., 1999). No contexto da Engenharia de *software*, o processo de transformação é denominado desenvolvimento, *inputs* são os requisitos do usuário, *outputs* são as funcionalidades do *software* produzido e os recursos são os atributos ou fatores componentes necessários à realização do processamento os requisitos identificados.

O processo de produção de *softwares* é composto de atributos diversos e heterogêneos que tornam sua modelagem bastante complexa. Desta maneira, é necessário um estudo detalhado sobre suas características para que se determinar a influência dele sobre o produto e sobre os fatores de produção, isto é, sobre a eficiência e eficácia do processo produtivo.

Serão a seguir consideradas duas visões deste processo: as visões Boehm (Boehm et. alii, 1998) e de Albrecht (IFPUG, 1994). Boehm tenta relacionar fatores de produção como o tempo e o esforço de desenvolvimento com todas as características tanto do processo quanto do produto por ele identificadas, usando regressão estatística para produzir fórmulas que poder ser usadas para estimar projetos que ainda serão executados. Por outro lado, Albrecht tenta obter os fatores citados de maneira relativa à visão do usuário. Ou seja, para cada entidade desenvolvedora é aplicado o seu algoritmo e obtido um valor representante para o tamanho funcional do *software*. Este valor é usado em conjunto com uma linha básica de projetos previamente executados para produzir estimativas dos fatores produtivos de futuros projetos.

Tais visões mencionam atributos capazes de mapear os principais modelos de medição de *softwares* utilizados no mercado. Desta maneira, objetiva-se identificar uma ampla quantidade de fatores, atualmente utilizados, tal que se permita construir um modelo de estimação de métricas bastante genérico.

2.3.1 Visão do processo por Boehm

O modelo COCOMOII, desenvolvido por Boehm, é um dos mais utilizados na representação do processo de construção, melhoria e manutenção de *softwares*. Ele considera que seu modelo construtivo de estimação de custo é composto por três sub-modelos:

composição de aplicação – utilizado para a estimação de esforço em projetos que se valem de ferramentas CASE; **projeto inicial** – que se baseia em informações não totalmente definidas para realização de estimativas; e **pós-arquitetura** – onde todo o projeto encontra-se definido e detalhado, conhecendo-se previamente sua arquitetura (Chulani et. alii., 1998).

O sub-modelo **pós-arquitetura** foi projetado para representar os programas e seu processo de desenvolvimento através de uma grande quantidade de fatores. Através do uso de tabelas pré-definidas, COCOMOII determina três constantes que tentarão representar o produto e seu processo. A métrica de esforço (*E*) é obtida aplicando-se a Eq. 2.1:

$$E = A \cdot q^{B} \cdot C \qquad (2.1)$$

A constante ***A***, em versões anteriores do modelo de Boehm, designava o tamanho do projeto, categorizado em três níveis: *orgânico*, *semidestacado* e *embutido*, como visto na Tabela 2.1. Em COCOMOII, tal constante e todas as outras são definida por uma combinação de estimação empírica e técnicas estatísticas. Chulani et. alii. (1998) oferece uma calibração ao parâmetro A com valor de **2.45**.

Modelo Parâmetro	Orgânico	Semidestacado	Detalhado
A	2.4	3.0	3.6

Tabela 2.1 – Parâmetro A *de acordo com o tamanho do projeto.*

A constante B representa o fator de escala do projeto e é determinado na Tabela 2.2, onde as linhas representam os fatores de escala e as colunas a graduação desses fatores. A precedência (PREC) é um fator que indica o grau de similaridade do produto com outros já desenvolvidos na entidade desenvolvedora; a flexibilidade (FLEX) representa o grau em que as restrições impostas pelos requisitos do produto influem sobre seu desenvolvimento; a arquitetura (RESL) visa indicar a eficácia da estrutura definida para o produto; a coesão da equipe (TEAM) mede o entrosamento da equipe, dificuldades de relacionamento, sincronização de seus componentes; a maturidade do processo (PMAT) é um fator que mede o grau de maturidade da entidade desenvolvedora, baseado no modelo CMM (Capability Maturity Model) produzido pelo SEI (Bade et. alii, 1995).

Fator de Escala	Muito Baixo	Baixo	Nominal	Alto	Muito Alto	Externamente Alto
Precedência	0.0405	0.0324	0.0243	0.0162	0.0081	0.0000
Flexibilidade	0.0607	0.0486	0.0364	0.0243	0.0121	0.0000
Arquitetura	0.0422	0.0338	0.0253	0.0169	0.0084	0.0000
Coesão da Equipe	0.0494	0.0395	0.0297	0.0198	0.0099	0.0000
Maturidade do Processo	0.0454	0.0364	0.0273	0.0182	0.0091	0.0000

Tabela 2.2 – Parâmetro B por classificação de fatores de escala.

Por último, a constante C tem seu valor definido conforme a Eq. 2.2, onde M_i representa a influência de cada direcionador de custo sobre o esforço de produção, fornecidos através de avaliações representadas por valores lingüísticos associados a ponderações pelo valor mais representativo de sua classe (Boehm, 1998).

$$C = \prod_{i=1}^{17} M_i \qquad (2.2)$$

Direcionador de custo	Muito Baixo	Baixo	Nominal	Alto	Muito Alto	Externamente Alto
DATA		0.93	1.00	1.09	1.19	
CPLX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
DOCU	0.89	0.95	1.00	1.06	1.13	
TIME			1.00	1.11	1.31	1.67
STOR			1.00	1.06	1.21	1.57
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.50	1.22	1.00	0.83	0.67	
PCAP	1.37	1.16	1.00	0.87	0.74	
AEXP	1.22	1.10	1.00	0.89	0.81	
PEXP	1.25	1.12	1.00	0.88	0.81	
LTEX	1.22	1.10	1.00	0.91	0.84	
PCON	1.24	1.10	1.00	0.92	0.84	
TOOL	1.24	1.12	1.00	0.86	0.72	
SITE	1.25	1.10	1.00	0.92	0.84	0.78
SCED	1.29	1.10	1.00	1.00	1.00	
RELY	0.75	0.88	1.00	1.15	1.39	

Tabela 2.3 – Direcionadores de custo.

São 17 (dezessete) os direcionadores de custo que influenciam as estimativas do processo de engenharia de *software*, como descritos abaixo. A Tabela 2.3 define seus valores (Chulani et. alii., 1998).

(a) DATA - Tamanho da base de dados: representa o efeito do tamanho das requisições de dados sobre desenvolvimento do produto;

(b) CPLX - Complexidade do produto: baseado em cinco classes de operações (controle, computacionais, dependente de dispositivos, gerenciamento de dados e interface com o usuário), esta medida busca mapear a complexidade do produto;

(c) RUSE - Reutilização: visa fornecer o grau de esforço adicional necessário à elaboração de projetos que serão compostos por componentes de *software* reutilizados em futuros projetos;

(d) DOCU - Documentação: leva-se em conta a influência da documentação do projeto sobre o esforço do processo;

(e) TIME - Tempo: mede a restrição na quantidade tempo de execução do *software*;

(f) STOR - Armazenamento: mede a restrição na quantidade de utilização de memória principal pelo *software*;

(g) PVOL - Plataforma: busca avaliar a volatilidade da plataforma, isto é, a quantidade de recursos de *hardware* e *software* que serão necessários ao funcionamento do sistema;

(h) ACAP - Capacidade do Analista: mede a habilidade, a eficiência, a visão e o grau de comunicação e cooperação do responsável pela elaboração da solução em *software* aos requisitos do usuário;

(i) PCAP - Capacidade do Programador: mede os mesmos indicadores utilizados na avaliação do Analista, mas com relação ao Programador;

(j) AEXP - Experiência com Aplicações: mapeia a experiência da equipe com o desenvolvimento de um específico tipo de aplicação;

(k) PEXP - Experiência com a Plataforma: leva em conta a experiência da equipe com os recursos disponibilizados pela plataforma na qual se baseia a aplicação, isto é, com os *softwares* e com os recursos disponíveis e utilizados na solução apresentada no projeto;

(l) LTEX - Experiência com a Linguagem e suas Ferramentas: mede a experiência da equipe com a linguagem e seu ambiente de desenvolvimento;

(m) PCON - Continuidade: expressa a taxa de rotatividade do pessoal envolvido com o processo de desenvolvimento;

(n) TOOL - Ferramenta de desenvolvimento: mede o nível produtividade fornecido pelos recursos disponibilizados pela ferramenta de desenvolvimento utilizada;

(o) SITE - Localização da Equipe: captura o efeito da forma de organização geográfica da equipe;

(p) SCED - Restrição de Cronograma: captura o grau de aceleração imposto ao desenvolvimento realizado pela equipe;

(q) RELY – Confiabilidade: representa o grau de segurança, confiança requisitada pelo *software*.

Além do esforço de desenvolvimento, o tempo de desenvolvimento pode se utilizar dos fatores anteriormente definidos. Desta forma, utilizando-se E - o esforço estimado pela Eq. 2.1 (não utilizando o multiplicador SCED) e recalibrando o valor de A para **2.66**, pode-se obter o tempo de desenvolvimento pela Eq. 2.3. VSCD é a percentagem de compressão/expansão do cronograma de desenvolvimento.

$$T = A \cdot \frac{VSCD}{100} \cdot E^{0.33 + 0.2 \cdot \sum W_{ij}} \qquad (2.3)$$

Conforme mostrado, o modelo de Boehm leva em conta muitos fatores componentes do processo, como também o tamanho do *software* a ser desenvolvido. Ela considera que *softwares* de tamanhos iguais são produzidos em tempo e com esforços diferentes se seus fatores de produção são também diferenciados. Tais fatores são medidos pelos direcionadores de custo descritos.

A seguir, apresenta-se um modelo que não considera os aspectos tecnológicos constitutivos do processo de desenvolvimento, mas sim uma visão da funcionalidade do *software*.

2.3.2 Visão do processo por Albrecht.

A visão de Albrecht usa como premissa a pequena variação dos fatores de processo para relacionar as características construtivas dos *softwares* com suas medidas respectivas de tempo e esforço de desenvolvimento, taxa de produção de *software*, produtividade da equipe etc.

Para Albrecht, um programa pode ser composto por dois tipos de funções: de dados e/ou funções transacionais. As funções de dados podem ser: **Arquivo** Lógico Interno ou Arquivo de **Interface** Externa. As transacionais podem ser: **Entradas** Externas, **Saídas** Externas ou **Consultas**. Cada um dos tipos de funções mencionado é descrito abaixo:

- **Entradas (*E*):** também conhecida como entrada externa, é a quantidade de entradas que proporcionam dados distintos orientados à aplicação. Quantifica as transações recebidas de fora da aplicação com o objetivo de atualizar os arquivos lógicos internos;
- **Saídas (*S*):** mede as saídas externas que proporcionem informações orientadas à aplicação, tais como relatórios e mensagens em terminal de vídeo;
- **Consultas (*C*)**: totaliza as entradas *on-line* que resultam na geração de alguma resposta imediata na forma de saída, como consulta a saldo em um sistema bancário. É subdividida em dois grupos determinantes: a **consulta parte da entrada** e a **consulta parte da saída***;* tratados pelo modelo a ser proposto mais à frente como pedido e resposta, respectivamente;
- **Arquivos (*A*)**: grupo lógico de dados do ponto de vista do usuário cuja manutenção é feita na própria aplicação é contado e totalizado por este número;
- **Interfaces (*I*)**: todas as comunicações realizadas pela máquina, necessárias à troca de informações entre a aplicação e um outro sistema externo, são resumidas por esse número.

A Figura 2.1 permite a visualização das funções componentes relativas à aplicação. Ela também identifica as fronteiras entre os agentes componentes do modelo de Albrecht.

O **usuário** é o ator que interage com a aplicação. Pode ser uma pessoa, um dispositivo ou um outro *software*.

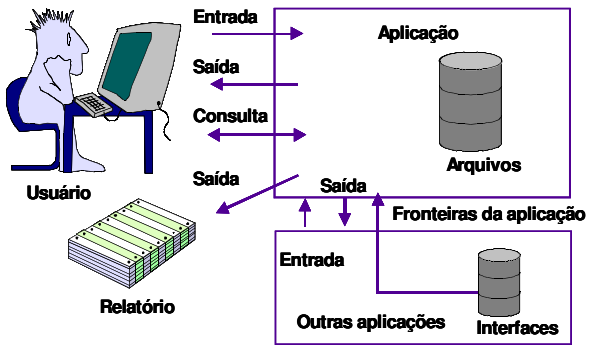

Figura 2.1 – Fronteiras da aplicação.

A **aplicação** é o *software* em avaliação cuja característica intrínseca é definida pelas funções Arquivos, Entrada, Saída e Consulta. É um conjunto de instruções, de dados e de procedimentos, associado a uma documentação operacional que o defina, que se destina a fornecer um específico conjunto de funcionalidades requisitadas pelo usuário.

Outras aplicações podem ajudar a aplicação em medição a fornecer a funcionalidade requisitada pelo usuário. São caracterizadas pelas funções Interfaces, Entrada e Saída.

Cada um desses tipos de função é caracterizado pelos seguintes atributos básicos: **itens de dados referenciados** – campos únicos não recursivos, identificados pelo usuário, que são atualizados em um arquivo lógico interno pela entrada externa (estão em todos os tipos de função); **registros lógicos** – subgrupo de dados reconhecido pelo usuário dentro de um arquivo de interface externa (estão nas funções de Arquivos e de Interfaces); **arquivos referenciados** – qualquer arquivo lógico interno consultado ou atualizado pelo processo ou qualquer arquivo de interface externa consultado (estão nas funções de Entradas, de Saídas e de Consultas).

A Análise de Pontos por Função (FPA) é um método desenvolvido em 1979 e publicado em 1984, por Allan Albrecht e, posteriormente, padronizado pelo IFPUG (International Function Point Users Group). É utilizado para a caracterização do produto (o *software*) a partir da visão do usuário. Segundo Goodbrand (1997), *a intenção de Albrecht foi inventar uma ferramenta que permitisse os projetistas estimarem o esforço requerido para completar uma tarefa, baseando-se nas funcionalidades dessa tarefa*.

Fixando-se a fase final do projeto lógico, no ciclo de vida clássico, como o marco para a realização de estimativas, FPA fornece uma métrica do tamanho do *software* relativa à sua funcionalidade medido em pontos de função.

Largamente utilizado, FPA é independente da tecnologia utilizada na elaboração do projeto ou na implementação deste, mas suas estimativas são ainda consideravelmente imprecisas, muito subjetivas e pouco claras (Pressman, 1995).

A Análise de Pontos por Função utiliza um modelo indireto de medição e tenta estabelecer um relacionamento entre a complexidade inerente ao processo com as características funcionais do produto. Ela tenta vencer as dificuldades associadas à predição

do esforço durante as atividades iniciais do ciclo de desenvolvimento (Longstreet, 2000; Lokan, 1999).

Para tanto, FPA baseia-se no fato que os programas constituem-se de funções compostas de unidades lógicas: dados e registros ou arquivos (componentes primários). De acordo com as quantidades de unidades lógicas, cada função é classificada quanto a sua complexidade e a esta é atribuído um peso determinado por experimentação. Os pesos das funções classificadas são acumulados por uma grandeza adimensional chamada **pontos de função** não ajustados. Finalmente, essa grandeza é "ajustada" pelo fator de influência das características gerais mencionadas.

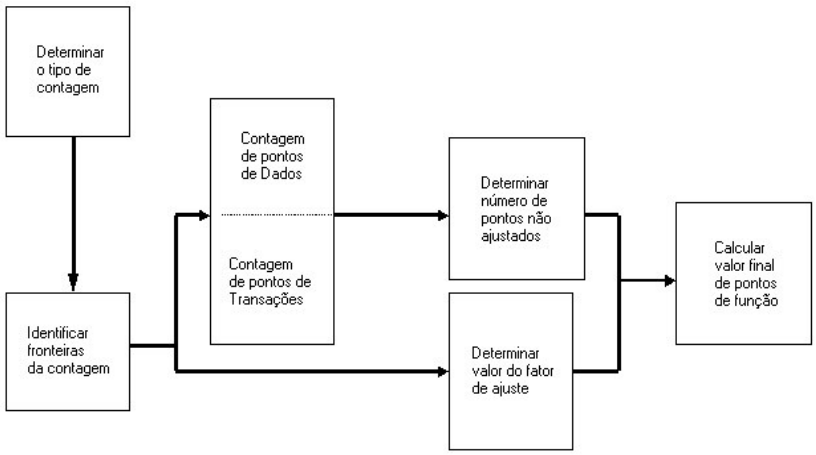

Figura 2.2 – Diagrama do processo de contagem de pontos de função.

A Figura 2.2 permite visualizar esquematicamente todo o processo de contagem de pontos de função de determinado projeto ou aplicação. Inicialmente, deve-se identificar o tipo de contagem, isto é, se é relativa ao dimensionamento de um projeto de desenvolvimento de um novo *software*; ou à manutenção de um *software*; ou ainda, à medição de uma aplicação já existente. São definidas, então, as fronteiras da aplicação, a partir de onde o processo de contagem se divide em duas tarefas: a medição dos aspectos específicos da aplicação e a medição do grau de influência dos aspectos gerais do sistema sobre a aplicação.

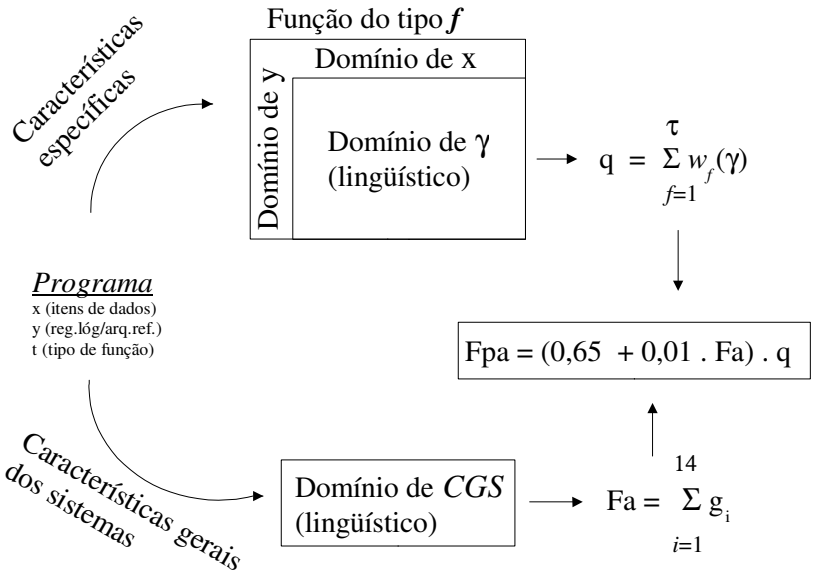

Figura 2.3 – Detalhamento do processo de contagem de pontos de função.

O manual SUNAT (2000) baseia-se na mais recente versão IFPUG de FPA. Nele pode-se ver que a aferição do tamanho do programa, conforme mostrado na Figura 2.3, ***Fpa*** consiste: na obtenção da quantidade de pontos de função (***q***), através de um processo de contagem; e na avaliação do grau de influência das características gerais dos sistemas sobre a funcionalidade total do ambiente sobre o programa (***Fa***). Realizados estes dois passos, determina-se o seu tamanho através da utilização da equação (2.4), cuja significação física é inexistente:

$$Fpa = \mathrm{q}.Fa \qquad (2.4)$$

O número de pontos de função (**q**) pode ser expresso em forma de equação como visto em (2.5), onde os valores $w_f(\gamma)$ são ponderações atribuídas à complexidade γ de cada função f requerida pelo usuário. T é o total de funções componentes do programa. Os tipos de função podem ser verificados na Figura 2.4.

$$\mathrm{q} = \sum_{f=1}^{T} w_f(\gamma) \qquad (2.5)$$

A complexidade γ, definida por Albrecht, é classificada pelo modelo de em Simples, Média ou Complexa, de acordo com a quantidade de itens de dados referenciados, registros lógicos e arquivos referenciados (KOKOL, 2000). Cada um desses valores lingüísticos de complexidade (Simples, Média, Complexa) é associado a um valor numérico (Figura 2.4), obtido empiricamente (Dekkers, 1998). Estes valores foram escolhidos buscando refletir o valor relativo da função na visão do usuário (Symons, 1988).

Tipo deFunção	Baixa	Média	Alta
EE	x 3	x 4	x 6
SE	x 4	x 5	x 7
CE	x 3	x 4	x 6
ALI	x 7	x 10	x 15
AIE	x 5	x 7	x 10

Arquivo Lógico Interno (ALI)
Arquivo de Interface Externa (AIE)
Entrada Externa (EE)
Saída Externa (SE)
Consulta Externa (CE)

Figura 2.4 – Ponderações para tipos de funções.

Para medir seu grau de influência sobre a funcionalidade geral do programa em avaliação, cada fator de ajuste de complexidade (resposta a pergunta contida na Tabela 2.4) é pontuado pelo avaliador em uma escala de 0 a 5, como visto na Figura 2.5:

1	O sistema requer *back*up e recuperação confiáveis?
2	São exigidas comunicações de dados?
3	Há funções de processamento distribuídas?
4	O desempenho é crítico?
5	O sistema funcionará num ambiente operacional existente, intensivamente utilizado?
6	O sistema requer entrada de dados *on-line*?
7	A entrada de dados *on-line* exige que a transação de entrada seja elaborada em múltiplas telas ou operações?
8	Os arquivos-mestres são atualizados *on-line*?
9	A entrada, a saída, os arquivos ou consultas são complexos?
10	O processo interno é complexo?
11	O código foi projetado de forma a ser reutilizável?
12	A conversão e a instalação estão incluídas no projeto?
13	O sistema é projetado para múltiplas instalações em diferentes organizações?
14	A aplicação é projetada de forma a facilitar mudanças e uso pelo usuário?

Tabela 2.4 – Computando valores de ajuste da complexidade.

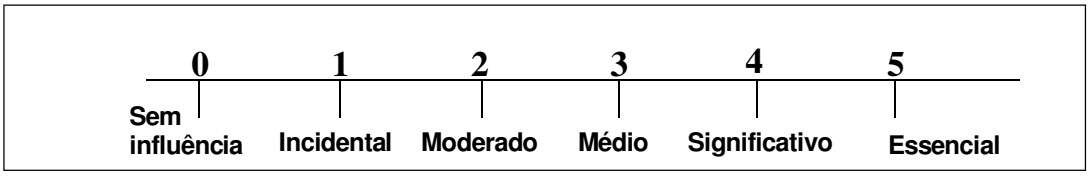

Figura 2.5 – Escala de graduação da influência de CGS sobre o programa.

Desta maneira, *CGS* pode ser obtido através da Equação (2.6), onde i é uma das 14 características existentes em um sistema e g_i o grau de influência dessa característica sobre o programa.

$$CGS = \sum_{i=1}^{14} g_i \qquad (2.6)$$

O fator de ajuste é obtido por:

$$Fa = 0{,}65 + 0{,}01.\sum_{i=1}^{14} g_i \qquad (2.7)$$

Da composição das equações (2.4), (2.5), (2.6) e (2.7), obtém-se o tamanho do programa dado pela Equação 2.8:

$$Fpa = \sum_{f=1}^{T} w_f(\gamma).(0{,}65 + 0{,}01.\sum_{i=1}^{14} g_i) \qquad (2.8)$$

Note-se que o fator de ajuste determinado pela Equação (2.7) pode influir sobre os pontos de função não-ajustados em até ±35% sobre seu valor nominal. O objetivo da realização deste ajuste de complexidade é aumentar o relacionamento da medida com estimadores de esforço, tentando capturar características não observadas pelos pontos de função não ajustados.

O uso do modelo de funcionalidade na visão do usuário, utilizado por FPA, é o responsável por tornar a métrica fornecida por este método (pontos de função ajustados) independente da tecnologia aplicada ao processo de desenvolvimento ou de manutenção. Esta característica a torna um atraente estimador do tamanho de programas, pois pode ser usada nas fases iniciais do ciclo de vida do desenvolvimento, além de ser considerada de simples compreensão por usuários não-especializados (Wu, 1997).

Os pontos de função ajustados são usados com vários propósitos (Lokan, 1999):

- como fator de normalização, facilitando a análise de diferentes projetos por diferentes especialistas de maneira uniforme;
- como variável dependente em modelos que objetivam seu relacionamento com o esforço de desenvolvimento;
- como unidade de conversão entre diferentes métricas de programa.

Entretanto, FPA é adequada a somente parte do domínio dos programas existentes, os sistemas de gerenciamento de informações (SGI), pois devido a sua grande granulosidade, subestima projetos no domínio do tempo-real (St-Pierre et. alii, 1997; Oligny et. alii, 1999). Além disso, tentando estabelecer um mapeamento entre complexidade e tamanho total das funcionalidades entregues ao usuário, fundamentando-se no esforço de desenvolvimento envolvido, o modelo em que se baseia a FPA é composto por várias escalas de transformação implícitas, algumas matematicamente inválidas (Abran e Robillard, 1996).

A uniformidade objetivada pelo método FPA é comprometida pela utilização de fatores subjetivos (*CGS*) que possibilitam inúmeras interpretações e dificultam a obtenção de resultados consistentes (VanDoren, 2001). Baseando-se em modelos de regressão Abran e Robillard (1996) afirma que a utilização de *CGS* não incrementa positivamente o relacionamento mencionado. Lokan (1999) chega a sugerir que da maneira que está definido, as *CGS* não deveriam ser usadas para o ajuste do tamanho do *software*, pois não consegue incrementar a qualidade da métrica em relação ao esforço e produtividade envolvidos. Em seu estudo, afirma ainda que talvez somente algumas características fossem relevantes e que algumas delas são mais importantes que outras (para diferentes projetos) e que elas pouco variam.

Devido às desvantagens mencionadas, foi propiciado um ambiente para o aparecimento de diversos refinamentos e variações de FPA (MKII, Feature points, COCOMO 2.0, FFP etc.). Este fato que levou a ISO (International Standards Organization) a promover um estudo, iniciado em 1993, para o desenvolvimento de um padrão de medição funcional, chamado Functional Size Measurement (FSM) sem considerar ajustes de complexidade (VanDoren, 2001).

O capítulo seguinte apresenta uma nova proposta de modelo que utiliza uma visão mista, tenta considerar todos os fatores, orientações, diretrizes e experiência até este ponto descrita.

CAPÍTULO 3

UM MODELO PARA ESTIMAÇÃO NA ENGENHARIA DE *SOFTWARES*.

De posse dos requisitos necessários à criação de um modelo que possibilite a realização de estimativas de métricas de *software*, será agora descrito o modelo em proposição neste trabalho. Os fatores anteriormente identificados serão agrupados conforme suas particularidades. Em seguida é apresentada uma representação do processo de desenvolvimento a partir da visão de Büren e Koll (1999).

3.1 Fatores característicos da engenharia de *softwares*.

Dentre os modelos utilizados na Engenharia de Software, mostradas anteriormente, podem-se identificar duas categorias de fatores componentes: **produto** e **processo**. Fatores relativos ao **produto** seriam aqueles que estão relacionados com alguma característica intrínseca dos *softwares*, como, por exemplo, seus componentes básicos. Fatores relativos ao **processo** estariam relacionados com características que definem o comportamento da entidade desenvolvedora durante a criação do *software*.

3.2.1 Fatores relativos ao produto – *software* (*q*)

O produto *software* carrega em si todo o resultado do processamento a que foi submetido. Isto é, à medida que vai sofrendo efeitos do processo de desenvolvimento, o *software* vai alterando sua utilidade, na visão do usuário (Albretch, 1979). Nesta ótica, as funcionalidades dos *softwares* relacionam-se com características como: tamanho, complexidade, confiabilidade, documentação, flexibilidade etc.

O *software* é usado em todos os modelos de estimação. Dentre eles, o modelo criado por Albrecht desconsidera todos os outros fatores, considerando apenas o *software* como determinante das medidas de processo, o que, como se viu anteriormente, é chamado independência da tecnologia (independência das características do processo). Na realidade, no modelamento proposto, Albrecht fornece uma medida de tamanho de *software* relativamente determinada em um conjunto de características tecnológicas pouco variáveis do processo. Ele admite que, internamente, uma entidade desenvolvedora deveria possuir um comportamento

padronizado, conhecido através de informações historicamente registradas. A partir daí, a construção de estimativas do processo seria uma simples tarefa de analogias.

Este trabalho considerará a hipótese de que os atributos básicos dos *softwares* estão relacionados com as métricas de produtividade de seu processo de construção. Ou seja, deve existir uma função que defina um comportamento esperado da produtividade tendo como variáveis independentes as características essenciais dos sistemas computacionais, considerando-se para tanto que as outras características pouco ou nada variem.

Como atributos básicos, serão considerados aqueles definidos por FPA: **itens de dados referenciados**, **registros lógicos** e **arquivos referenciados**.

3.2.2 Fatores relativos ao processo

No contexto da Teoria Geral da Administração, Chiavenato (2001) realizou um estudo a respeito dos componentes básicos de uma organização: **tarefas**, **estrutura**, **pessoas**, **ambiente** e **tecnologia**. Para ele, *o comportamento desses componentes é sistêmico e complexo: cada qual influencia e é influenciado pelos outros componentes. Modificações em um provocam modificações em maior ou menor grau nos demais*.

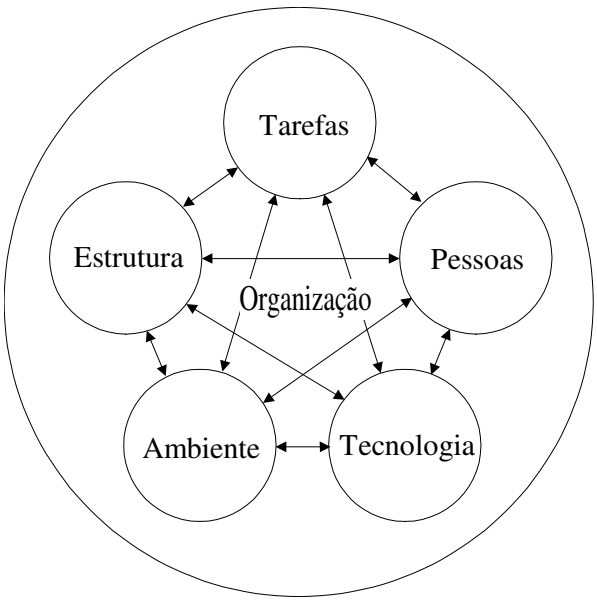

Figura 3.1 – Variáveis básicas na Teoria Geral da Administração.

Note-se que (Figura 3.1) o produto elaborado pela organização de Chiavenato não é considerado, pois retrata apenas os componentes do processo realizado. Fazendo-se uma

analogia entre tal organização de Chiavenato e uma entidade desenvolvedora de *softwares*, anteriormente definida, pode-se verificar a seguinte similaridade:

- **tarefas** e **metodologia** – conjunto de tarefas a serem realizadas;
- **estrutura** e **organização** – organização administrativa estabelecida pela entidade;
- **ambiente** e **plataforma** – plataforma de *hardware* e/ou de *software* para qual será desenvolvido o *software*;
- **tecnologia** e **ferramenta** – grupo de artefatos tecnológicos que compõem as ferramentas utilizadas;
- **pessoas** e **equipe** – grupo de pessoas que participaram da construção do *software*. Veja-se, a seguir, maior detalhamento a respeito de cada um desses fatores.

a) *Ferramenta – (**f**)*

Este fator indica **com o que** deverá ser construído o *software*. É definido, em muitos modelos de estimação, pelas técnicas utilizadas na implementação do produto do processo de desenvolvimento. Elas exercem forte influência sobre o tempo e o esforço de desenvolvimento.

São ferramentas de desenvolvimento qualquer artefato de *hardware* ou de *software* que automatize uma rotina existente no processo da engenharia de *softwares*. A linguagem de programação, as ferramentas CASE, as técnicas de remoção de erros e os recursos de codificação, são exemplos dessas ferramentas.

b) *Metodologia – (m)*

A metodologia é o fator do processo que indica **como** se deveriam realizar as tarefas de construção do *software*. Ela se baseia na padronização de tarefas que poderiam ser executadas de diferentes formas. Trata-se de um conjunto de algoritmos previamente estabelecidos que levam à automatização, para que se possibilite uma maior produtividade.

Este fator é responsável pelo combate à complexidade da solução apresentada ao usuário em forma de *software*. A metodologia leva em conta, por exemplo, o grau de reutilização de pacotes já implementados e o grau de sistematização oferecido pela metodologia.

c) *Plataforma – (p)*

É o fator que determina **onde** deverá ser construído o *software*, baseando-se nos recursos de *hardware* e *software* utilizados, bem como nas restrições impostas por cada um desses recursos. Está relacionado à arquitetura de *hardware*, ao sistema operacional, ao banco de dados etc., à tecnologia básica utilizada na construção do *software* como um todo.

Softwares que funcionem em diferentes plataformas possuem uma característica denominada de **portabilidade**. Este fator deve registrar o esforço na direção da padronização das funcionalidades oferecidas pelos recursos básicos utilizados na execução do *software* tanto pelo desenvolvedor quanto pelo usuário.

d) *Organização – (o)*

Este fator caracteriza a influência que a forma de organização administrativa da entidade desenvolvedora exerce sobre o processo de desenvolvimento. Mais precisamente, deverá representar a influência da organização sobre os recursos humanos disponíveis nessa entidade e a forma de processo de produção.

Ele pretende evidenciar formas de controlar as diferentes formas organizacionais disponíveis tal que se possa gerenciar o processo de engenharia de *software* mais eficientemente. Participam desta categoria os aspectos organizacionais dos recursos como: as **circunstâncias geográficas,** a **organização,** a **continuidade**, a **localização**, a **coesão** da equipe desenvolvedora; e a **maturidade** do processo.

e) *Equipe – (e)*

Talvez o mais importante dos fatores, devido a sua grande influência sobre o processo de engenharia de *softwares*, o fator equipe agrega informações sobre o pessoal responsável pela execução do trabalho de desenvolvimento. Informações como sua experiência, sua quantidade de componentes, seu rendimento (satisfação, remuneração, por exemplo), são alguns dos atributos considerados em sua definição.

Estão nesta categoria itens como: a **experiência da equipe**, a **capacidade do analista**, a **capacidade do programador**, a **experiência com aplicações**, a **experiência com a plataforma**, a **experiência com a linguagem e com suas ferramentas** e a **precedência**, da solução do problema do domínio.

3.2 Esforço de desenvolvimento na visão de Büren e Koll.

Büren e Koll (1999) construíram um diagrama de blocos representativo do esforço dispendido no processo de desenvolvimento, onde foi assinalado cada um dos fatores produtivos que participam da construção do *software*. A Figura 3.2 mostra a representação gráfica do modelamento do processo em questão.

O **diagrama de tarefas** representa as atividades centrais do processamento dispostas em um cronograma de tempo. Ele recebe informações que descrevem: o **produto** a ser construído (definido pelos seus requisitos funcionais e não-funcionais); o **problema** relativo ao domínio do sistema a ser representado pelo *software*; o **projeto** que descreve o que e como deverá ser construído o *software*; o **ambiente** no qual está inserido a entidade desenvolvedora; e o **processo** a ser executado através de um conjunto de **atividades** que utilizarão os recursos disponíveis para a construção do *software* requisitado.

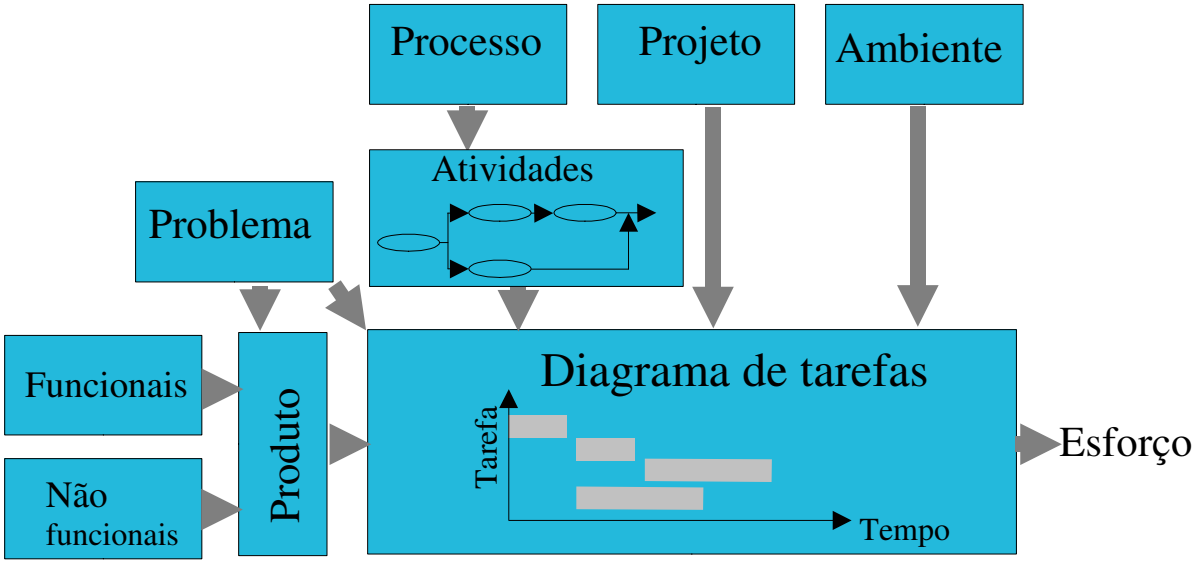

Figura 3.2 – Processo de desenvolvimento por Büren e Koll.

Da interação entre as informações processadas resulta um esforço de desenvolvimento (medido em pessoas.horas), ou seja, a quantidade de pessoas que participaram da construção durante um certo período de tempo, medido em horas.

Aproveitando a idéia do processo de construção de *softwares* passada pelo modelamento de Büren e Koll, o item que se segue proporá o modelo objetivado por esta dissertação.

3.3 Proposta de estimador de tempo e esforço de desenvolvimento

Considerados os fatores anteriormente identificados, em conjunto com o modelamento do processo produtivo proposto por Büren e Koll, pode-se construir um novo modelamento que possa ser utilizado na consecução do objetivo deste trabalho. O *software* é produto do processamento de informações (requisitos funcionais e não-funcionais) fornecidas pelo usuário a respeito do sistema real. Dependendo dos requisitos identificados no modelo do sistema este processamento será realizado em um maior ou menor tempo. O processamento de um mesmo conjunto de requisitos tem seu comportamento afetado pelas características da entidade desenvolvedora.

O modelo do sistema é caracterizado nesta proposta pelos atributos básicos dos *softwares* definidos por Albrecht. Diferentemente, no entanto, é o uso desses fatores. Considera-se que as quantidades totais de atributos, agrupados por tipo de função, são capazes de representar o software solicitado. Pro exemplo, para representar-se as entradas solicitadas pelo usuário, e considerada separadamente, a quantidade totoa de itens de dados de todas as funções do tipo entrada, e a quantidade total de registros lógicos, também, de todas as funções do tipo entrada.

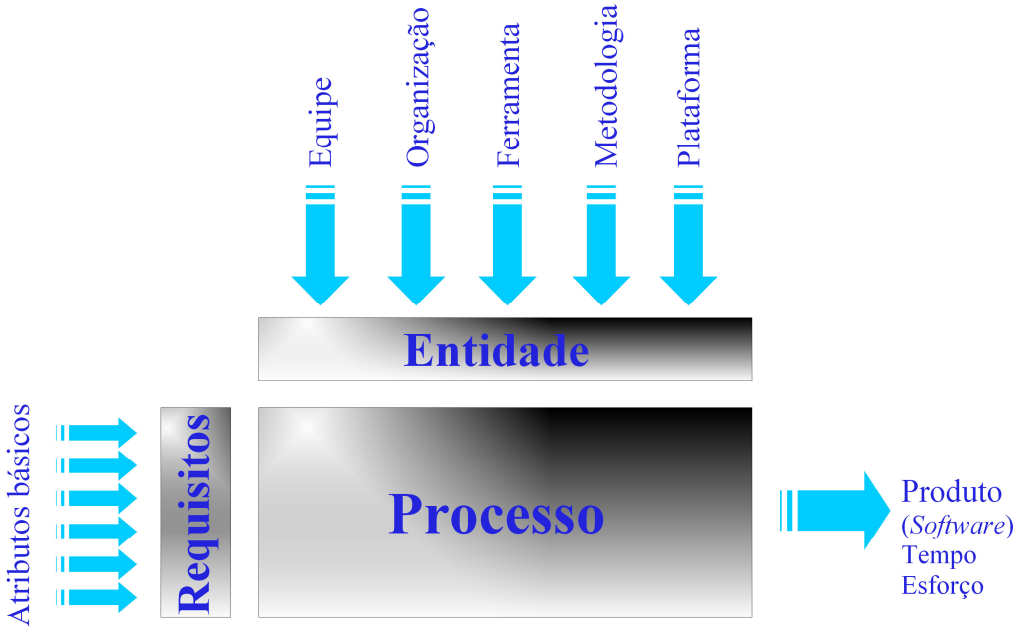

Figura 3.3 – Modelo para estimação de tempo e esforço de desenvolvimento.

O comportamento do processo é definido pelos fatores caracterizadores da entidade desenvolvedora. É aqui admitido que a equipe, a organização, a ferramenta a metodologia e a plataforma são capazes de representar o tipo de processo a que serão submetidos os requisitos do sistema. A Figura 3.3 retrata a visão do processo de desenvolvimento de *softwares* em proposição. O produto *software* resulta das ações efetuadas por uma entidade desenvolvedora sobre o modelo construído para representar um problema no domínio do usuário, representado pelos requisitos do sistema. Os fatores tecnológicos que identificam o processo definem o comportamento da entidade durante a construção desse produto.

A partir de um comportamento específico de uma entidade, os atributos básicos do *software* a ser construído podem determinar diretamente métricas (tempo, esforço) relativas ao processo desenvolvimento. Tais atributos básicos devem ser auferidos por tipo de função (Entrada, Saída, Consulta, Interface e Arquivo Lógico) de forma cumulativa para cada *software* em avaliação, como anteriormente mencionado. Desta maneira, a atividade de medição de fatores produtivos transforma-se em mero registro de quantidades de atributos que deverão existir no *software* que será construído.

O capítulo seguinte descreve a arquitetura de um *software* que implementa um protótipo para estimação de métricas de tempo e de esforço de desenvolvimento de *softwares*, a partir do modelo de construção de *softwares* ora proposto.

CAPÍTULO 4

UM ESTIMADOR DE TEMPO E ESFORÇO DE DESENVOLVIMENTO DE *SOFTWARES*.

Este capítulo descreve a arquitetura de um sistema que possibilita a estimação de tempo e esforço de desenvolvimento de *softwares*, tendo como base a representação oferecida pelo modelo proposto no Capítulo 3. Trata-se de um protótipo construído com um conjunto coordenado de objetos de *software* que interagem entre si para fornecer estimativas para as métricas de desenvolvimento de *software* mencionadas, utilizando informações a respeito da entidade produtiva: processo e produto.

Inicialmente, são mencionados alguns conceitos sobre o modelamento OO (Orientado a Objetos) e a (UML) Linguagem de Modelagem Unificada. Em seguida, o protótipo é especificado em UML, mostrando-se, através de diagramas de casos de uso e de classes, suas funcionalidades. Finalmente, é descrita a implementação do sistema desenvolvido.

4.1 Modelamento orientado a objetos

O método de desenvolvimento orientado a objetos baseia-se em uma abstração do mundo real com que o ser humano possui experiência desde suas mais incipientes formas de conhecimento: o objeto. Desde que tudo que se oferece à vista ou ao espírito pode ser visualizado por um objeto, este é entendido como uma instância de uma classe de elementos que possuem estrutura e comportamento semelhantes.

Um objeto de *software* é um elemento independente que representa um grupo de recursos inter-relacionados e projetado para realizar tarefas específicas. Para Price (1997) um objeto é *uma unidade dinâmica, composta por um estado interno privativo (estrutura de dados) e um comportamento (conjunto de operações).* Quando se fala em um sistema de objetos, está-se referenciando um conjunto conexo de objetos que trabalha integradamente para realizar tarefas predefinidas. Booch (1994) define programação orientada a objetos da seguinte forma:

> *Um modelo de programação em que programas são organizados como coleções cooperativas de objetos, em que cada um deles*

> *representa um exemplo de algum tipo, e cujos tipos são todos membros de uma hierarquia de tipos unidos uns aos outros através de um relacionamento de herança.*

Para Booch, um sistema pode ser considerado orientado a objetos se possuir todos os seguintes elementos: abstração, encapsulamento, modularidade e hierarquia.

Uma abstração denota características essenciais do objeto que o distingue dos outros tipos de objetos, onde são definidas precisamente suas fronteiras conceituais a partir da perspectiva do observador.

O encapsulamento é o processo de agrupamento de elementos de uma abstração que constitui a estrutura e comportamento desta abstração, servindo para separar sua implementação (parte interna) do mundo exterior. O acesso às características da abstração é limitado e definido através de uma interface de comunicação.

A modularidade é a propriedade que possui um sistema que mede o grau de decomposição em conjuntos coesos, fracamente acoplados e menores, chamados módulos. O conceito de coesão está intimamente ligado com o grau de atomicidade do módulo.

O último elemento essencial do modelamento orientado a objetos é a hierarquia, processo que classifica e ordena um conjunto de abstrações inter-relacionadas. A hierarquia pode se manifestar através de mecanismos como herança e agregação. No mecanismo de herança, ocorre um relacionamento através do compartilhamento da estrutura e do comportamento da abstração mais genérica com a abstração mais específica. Este mecanismo é também designado especialização/generalização. No mecanismo de agregação, há o relacionamento de pertinência entre duas diferenciadas abstrações.

4.2 Linguagem de modelamento unificada

Devido à necessidade de representação das abstrações acima, de forma eficiente e padronizada, a UML – linguagem de modelamento unificada foi criada através da composição de várias diferentes notações, dentre elas: OMT, Método de Booch e OOSE. UML pode ser utilizada para visualização, especificação, construção e documentação de *softwares*.

Na construção de um modelo de sistema de *software*, a linguagem UML se utiliza de múltiplas visões para produzir diagramas que representarão cada um de seus componentes,

tal que se tenha a maior clareza a seu respeito. Dentre esses diagramas, o de casos de uso e o de classes são essenciais à descrição de um sistema de *software*.

O diagrama de casos de uso tenta capturar a funcionalidade a partir da visão do usuário. É construído nos primeiros estágios do ciclo de desenvolvimento, suas metas são a especificação do contexto do sistema, captura dos requisitos do sistema, validação da arquitetura do sistema, direcionamento da implementação e geração de casos para testes.

O diagrama de classes busca capturar o vocabulário do sistema, sendo construído e refinado através da conceituação baseado no mapeamento de abstrações. Seu objetivo é dar nomes aos conceitos identificados pelas abstrações do sistema, identificar os relacionamentos existentes entre tais abstrações e especificar um esquema lógico de dados.

4.3 Especificação do sistema estimador de tempo e esforço de desenvolvimento de *softwares*.

O sistema em especificação foi responsabilizado pela produção de estimativas para métricas de tempo e de esforço de desenvolvimento, relativas a produtos de *software* construídos através de processos de engenharia realizados em entidades de desenvolvimento. Esta entidade é constituída por: equipe de produção, ferramenta de desenvolvimento, metodologia e plataforma, submetida a uma determinada forma de organização administrativa. Ela é a responsável pela produção de um produto de *software*, que será caracterizado pelo conjunto de atributos básicos, definidos anteriormente a partir da visão de Albrecht.

Dois módulos compõem o sistema em desenvolvimento: o módulo **Gerente** e o módulo **Especialista**. O módulo **Gerente** foi responsabilizado pela seleção da classe de processo que melhor representa o desenvolvimento de *software* corrente, através do recebimento das informações relativas aos componentes desse processo (f, m, p, o, e). Selecionada a classe da entidade, o módulo **Especialista** pode fornecer estimativas a respeito das métricas de desenvolvimento, considerando os apenas os componentes específicos constitutivos do produto de *software* em avaliação.

4.3.1 Diagramas de casos de uso

Como descrito anteriormente, o produto (*software*) pode ser definido por seus atributos básicos. A partir do conhecimento desses atributos, o módulo **Especialista**,

representado por uma rede neural *feed forward* treinada com algoritmo *back propagation* (Apêndice A) pode fornecer as estimativas das métricas tempo e esforço de desenvolvimento, relativas ao processo específico definido no módulo **Gerente**. Ele também pode fornecer a medida do tamanho do *software* em pontos de função não ajustados (IFPUG, 1994), isto é, sem levar em conta o fator de ajuste de complexidade.

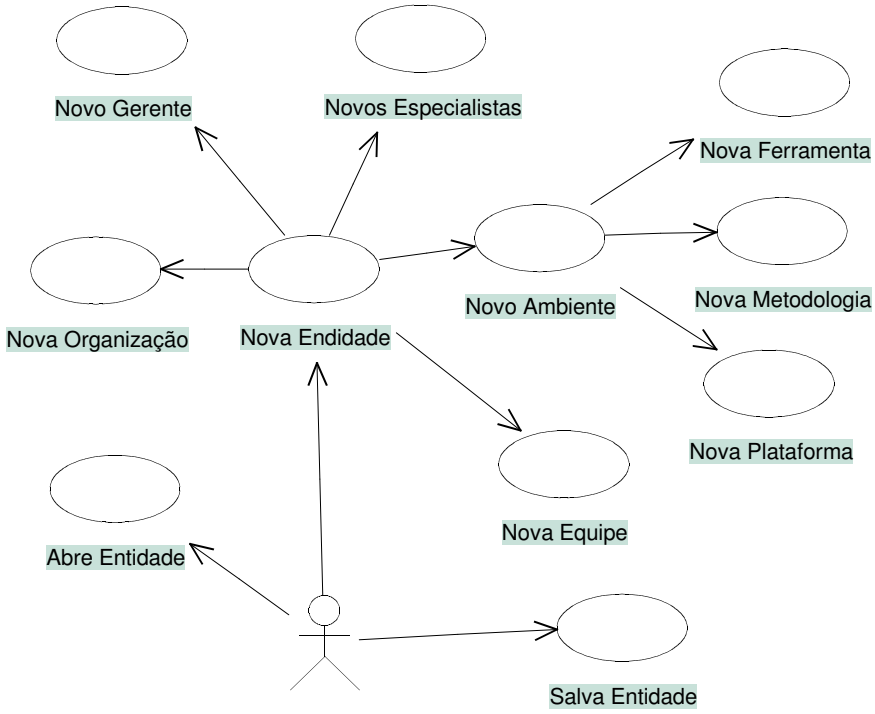

Figura 4.1 – Diagrama de casos de uso: Entidade.

A utilização do sistema parte da criação de uma nova **entidade** de desenvolvimento (menu *Entidade|Cria*), definida por um **ambiente** (**ferramenta**, **metodologia** e **plataforma**), uma **equipe** e uma **organização**. Criada a **entidade**, poder-se-á guardar as informações fornecidas através da opção de menu *Entidade|Salva*, para posterior recuperação através da opção de menu *Entidade|Abre*. O diagrama de casos de uso relativo à **Entidade** pode ser visto na Figura 4.1.

Em seguida, cria-se um **gerente** com capacidade de classificar uma quantidade pré-determinada de processos homogêneos (Figura 4.1). Isto é, considerando-se que existem muitas entidades desenvolvedoras, mas que estas podem-se agrupar em classes, de acordo com um conjunto de similaridades qualquer, o **gerente** deve ser capaz de classificar uma

entidade para ele desconhecida em uma classe de entidades cujos comportamentos sejam homogêneos. Para tanto, dever-se-á treinar o **gerente**, que, para cada processo homogêneo identificado, automaticamente, criará um **especialista**.

A quantidade de classes utilizadas pelo **gerente** será determinada após a indicação do tamanho do mapa definido pela rede neural utilizada (Apêndice A). Essa quantidade dependerá do grau de similaridade existente entres as entidades utilizadas durante o treinamento do **gerente**. Devido a não se dispor de bastantes dados sobre projetos reais de desenvolvimento já executados, no treinamento do **gerente**, serão utilizados os direcionadores de custo de Boehm (1998), vistos no Capítulo 2, para a geração dos padrões de entrada.

Responsável pelas estimativas de tempo e de esforço de desenvolvimento, o **especialista** deverá também ser treinado (Figura 4.2) em cada classe de processo homogeneamente caracterizado, baseando-se na utilização dos atributos básicos dos *softwares* (conceituados a partir da visão de Albrecht).

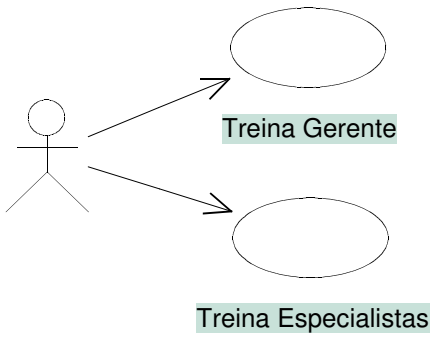

Figura 4.2 – Diagrama de casos de uso: Treinamento.

A caracterização da entidade de desenvolvimento poderá ser alterada através da opção de menu: *Configura*. Quando se usar a opção *Entidade*|*Salva*, serão guardadas as informações contidas nessa configuração. Na opção de configuração, serão ajustados: o gerente, definindo-se o seu mapa de características, os respectivos especialistas que serão usados pelo gerente, a organização da entidade, a Equipe de desenvolvimento e o ambiente de produção de *software*, composto de uma ferramenta, uma metodologia e uma plataforma. A Figura 4.3 mostra o diagrama de casos de uso de configuração.

Configurada a **entidade**, poder-se-á criar um novo projeto de *software* através da opção de menu *Programa|Novo*. Um **programa** é definido no sistema como um conjunto de funções agrupadas por tipo e referidas por seus atributos básicos. Poder-se-á guardar as informações fornecidas sobre o *software* utilizando-se a opção de menu *Programa|Salva*, para posterior reutilização, que ocorreria pelo uso da opção de menu *Programa|Abre*.

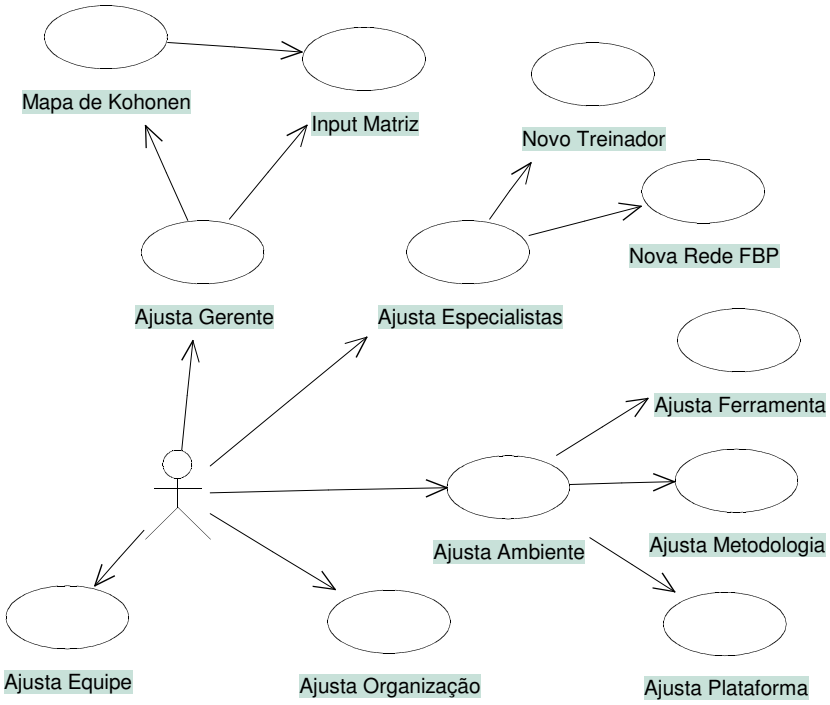

Figura 4.3 – Diagrama de casos de uso: Configuração.

Fornecidas as informações da **entidade** e do **programa**, poder-se-á: medir o tamanho do *software* corrente em pontos de função pela utilização da opção de menu *Programa|Tamanho*; e estimar métricas de tempo e de esforço de desenvolvimento do *software* corrente através da utilização das opções de menu *Programa|Estima|Tempo* e *Programa|Estima|Esforço*, respectivamente.

As opções de menu *Ajuda|Conteúdo* e *Ajuda|Sobre* fornecem dados relacionados com o sistema. A Figura 4.4 permite visualizar-se o diagrama de casos de uso relativo a **Programas**.

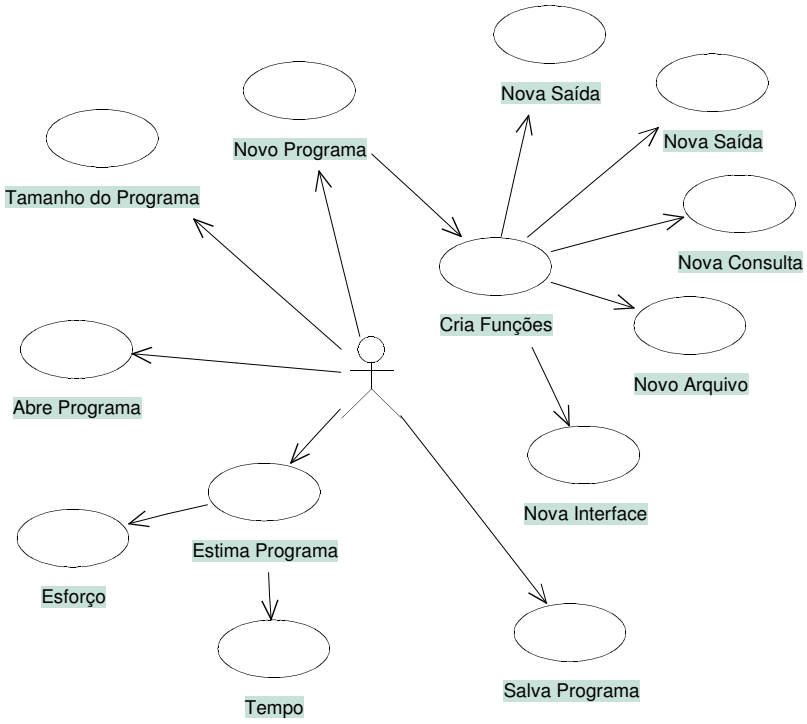

Figura 4.4 – Diagrama de casos de uso: Programa.

4.3.2 Arquitetura de classes do sistema

O módulo **Gerente** do processo é composto pelas classes **KohonenFeatureMap** e **InputMatrix**, do pacote **jfröhlich.NeuralNets** (Apêndice C). Este módulo também usa as classes **Equipe**, **Especialista**, **Organização** e **Ambiente**. A classe **Ambiente** é composta de objetos das classes **Metodologia**, **Ferramenta** e **Plataforma**. A partir da utilização da visão de Boehm, a classe **Equipe** será caracterizada pelos atributos ACAP, PCAP, AEXP, PCON, PREC, TEAM e LTEX. A classe **Organização** possuirá os seguintes atributos: PMAT e SITE. As classes **Metodologia** e **Ferramenta** serão definidas por um único atributo cada uma: RUSE e TOOL, respectivamente. **Plataforma** será uma classe que possuirá os seguintes atributos: TIME, STOR e PVOL. A Figura 4.5 mostra o diagrama de classes deste módulo.

A classe **Especialista** utiliza a classe ***feedforward*BP**, definida a partir do pacote **jahuwaldt.NeuralNets**. Este pacote de simulação de redes neurais, conforme descreve Huwaldt (1999), foi projetado tal que possa ser rápida e facilmente modificado. Pode-se criar uma nova arquitetura de rede ou usar uma já criada. Suas principais classes são abaixo descritas.

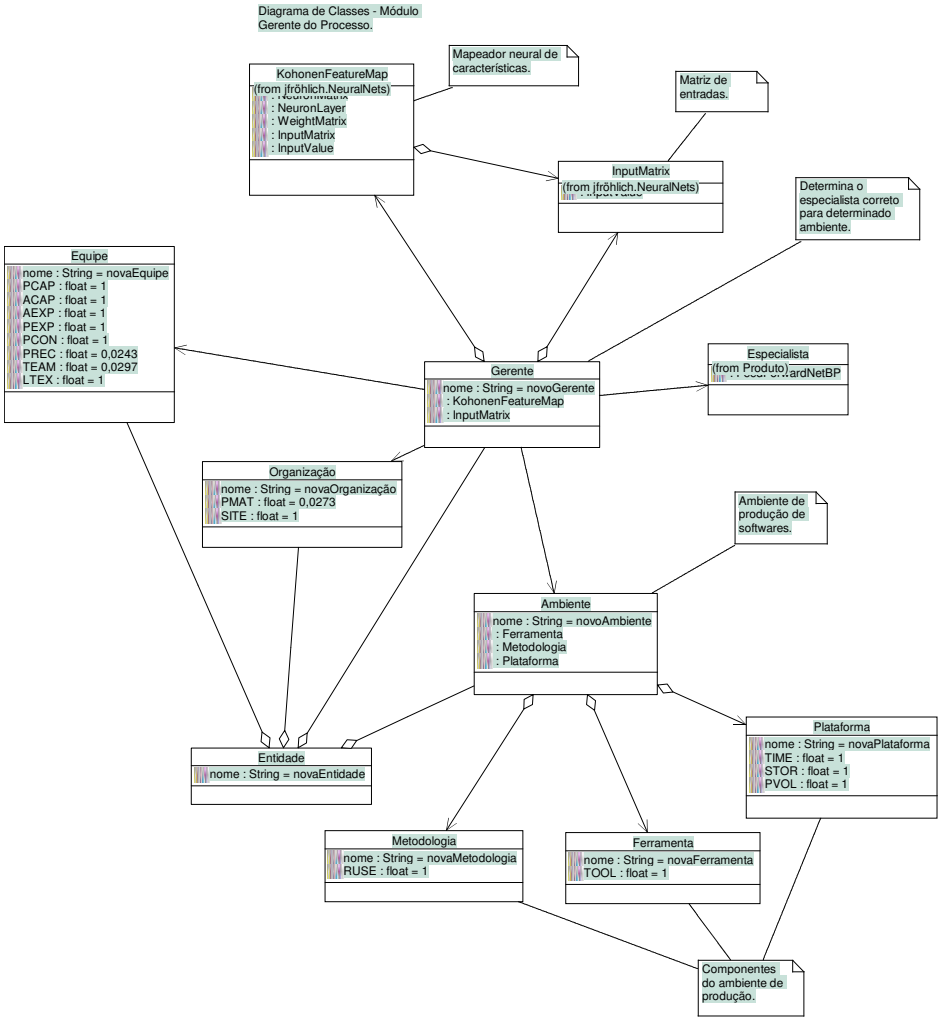

Figura 4.5 – Diagrama de classes do módulo Gerente.

A classe ***feedforward*** implementa um padrão multicamadas para uma rede com arquitetura *feed forward*. Nela, não se especifica o método de aprendizagem e assim pode-se facilmente estender esta classe para implementar qualquer tipo de aprendizado em qualquer neurônio.

O módulo **Especialista**, relacionado ao produto, será composto por um objeto da classe ***feedforward*NetMap** que fornecerá uma rede neural *feed forward* treinada com um algoritmo *back propagation*. Um objeto da classe **TrInstGenerator** será utilizado para realização do treinamento da rede neural *feed forward* criada com os padrões entrada-saída fornecidos. A classe ***feedforward*BP** provê uma simples implementação do algoritmo de aprendizado *back propagation*.

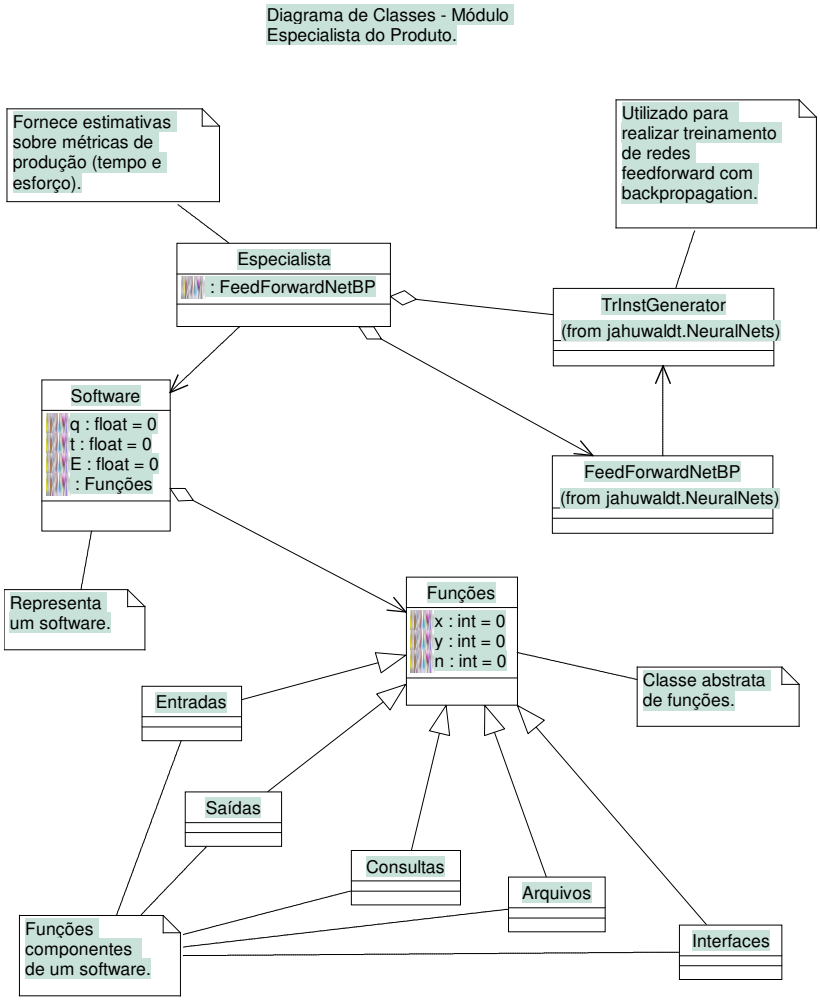

Figura 4.6 – Módulo Especialista.

A classe **Especialista** relaciona-se com a classe ***software*** através de um relacionamento de agregação. ***software*** é composta dos atributos: ***q***; tamanho do *software*; ***t***, tempo de desenvolvimento estimado; ***E*** e um conjunto de funções definidas a partir da visão de Albrecht. Este conjunto de funções é definido pelos atributos básicos do *software*, referenciados através das variáveis: ***x***, quantidade de itens de dados; ***y***, número de arquivos referenciados ou número de arquivos lógicos internos; e ***n***, quantidade de funções, todas relacionadas ao tipo de função utilizada (Entrada, Saída, Consulta, Arquivos e Interface). A Figura 4.6 mostra o diagrama de classes do módulo **Especialista**.

4.4 Implementação do especialista estimador.

Do sistema de estimação especificado, foram implementados dois programas. O primeiro visou o módulo **Especialista**, com o fim de testar o possível relacionamento entre os

atributos básicos do *software* e o tempo de desenvolvimento de *softwares*. Para avaliação dessa hipótese, foram utilizados dados de projetos executados (Apêndice B) contidos no trabalho de Abran (1996).

O segundo programa implementa o protótipo do sistema de estimação do tempo e do esforço de desenvolvimento especificado nos itens anteriores.

4.4.1 Relacionamento dos atributos básicos dos *software* e seu correspondente tempo de desenvolvimento.

FPA busca relacionar a funcionalidade total do programa entregue ao usuário e o esforço empregado em seu desenvolvimento. Em seu trabalho, Albran (1996) realizou um estudo empírico da metodologia de Albrecht, através de análises estatísticas de um conjunto de dados contendo vinte e um projetos (vide Apêndice B), cujos fatores de influência sobre a produtividade foram considerados homogêneos, isto é, todos os componentes do processo mantiveram-se invariáveis durante a produção dos vários projetos. A partir daí, Albran propôs dez modelos baseados em componentes primários e em componentes complexos do programa.

Os melhores resultados obtidos por Albran foram com o modelo com 10 variáveis, onde consideraram-se os atributos básicos, e com o modelo com 5 variáveis, onde consideraram-se os valores em pontos de função relativos a cada grupo de funções.

Considerando os dados fornecidos por Albran (1996) sobre projetos de sistemas executados, foi implementado um sistema que se propõe implementar o modelo proposto, no contexto especialista. Isto é, considerando que pouca variação ocorra nos componentes da entidade desenvolvedora, o sistema poderia capturar diretamente a relação existente entre os componentes primários do *software* e seu correspondente tempo de desenvolvimento.

A Figura 4.7 permite visualizar um diagrama semelhante ao utilizado para representar o modelo produzido (o primeiro programa produzido não utilizou o esforço de desenvolvimento como saída). Como se pode verificar, trata-se da representação de uma rede neural artificial, cuja arquitetura tenta relacionar as quantidades totais de cada atributo existentes em cada tipo de função solicitada pelo usuário com o correspondente tempo e esforço de desenvolvimento do *software* em avaliação.

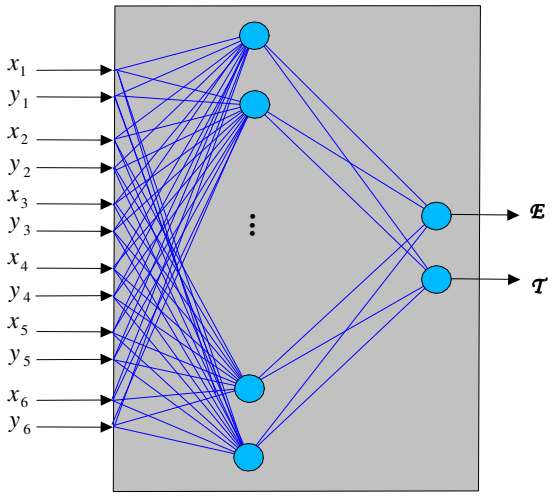

Figura 4.7 – Modelo baseado em rede neural artificial.

A rede construída é composta de três camadas: **entrada** – para 12 (doze) variáveis, (x_i, y_i) com $i \in (1,2,...6)$ representam os atributos básicos dos *softwares*. $\mathcal{E}$ e $\mathcal{T}$ representam respectivamente o esforço e o tempo dispendido durante o desenvolvimento do *software*.; **oculta** – quantidade de neurônios a ser ajustada pela experimentação; e **saída** – para 02 (duas) variáveis. Ela utilizou a arquitetura *feed forward*, treinada com aprendizado *back propagation*. Seus parâmetros variáveis foram ajustados por valores reais dos tempos despendidos (dado em horas) e do esforço (pessoas.horas) na construção e integração das funções componentes do programa e seus respectivos pontos de função associados.

Cada tipo de função componente (entradas, saídas, consultas, arquivos, interfaces) é caracterizado por suas quantidades totais de componentes primários, definidas pela metodologia de Albrecht, como seguem descritas:

- **Camada de entrada:** x_1 – número de itens de dados da função de entrada externa; y_1 – número de arquivos referenciados da função entrada externa; x_2 – número de itens de dados da função de saída externa; y_2 – número de arquivos referenciados da função entrada externa; x_3 – número de itens de dados da função de consulta parte da entrada; y_3 – número de arquivos referenciados da função consulta parte da entrada; x_4 – número de itens de dados da função de consulta parte da saída; y_4 – número de arquivos referenciados da função consulta parte da saída; x_5 – número de itens de dados da função arquivo lógico interno; y_5 – número de registros lógicos da função arquivo lógico interno; x_6 –

número de itens de dados da função arquivo de interface externa; e y_6 – número registros lógicos da função arquivo de interface externa;

- **Camada de saída**: $\mathcal{T}$ – tempo de desenvolvimento, medido em *horas*; $\mathcal{E}$ – esforço de desenvolvimento, dado *pessoas.horas*.

Baseado na linguagem Java, foi criada uma interface (Figura 4.8) que possibilita a criação de uma RNA/BP (rede neural *feed forward* treinada com algoritmo *back propagation*) para a estimação do tempo de desenvolvimento de *softwares*. Foi possibilitado proceder-se o ajuste dos parâmetros da rede de forma que se possuísse flexibilidade para testes. Inicialmente, realizou-se o treinamento de RNA/BP com um conjunto de dados contendo 100% dos projetos fornecidos por Albran e Robillard (Apêndice B). Através dos testes realizados, constatou-se que os projetos, identificados por IDAP: 4, 6, 28, 29 e 44, forneceram desvios-padrão muito acima da média do conjunto.

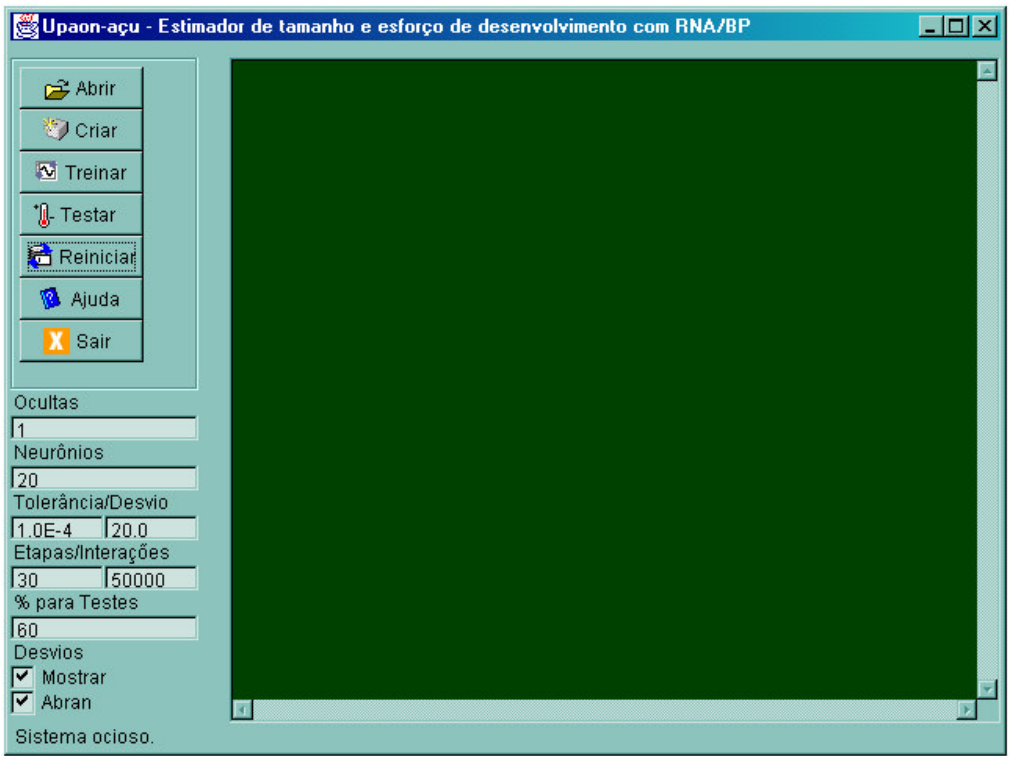

Figura 4.8 – Sistema especialista em produto.

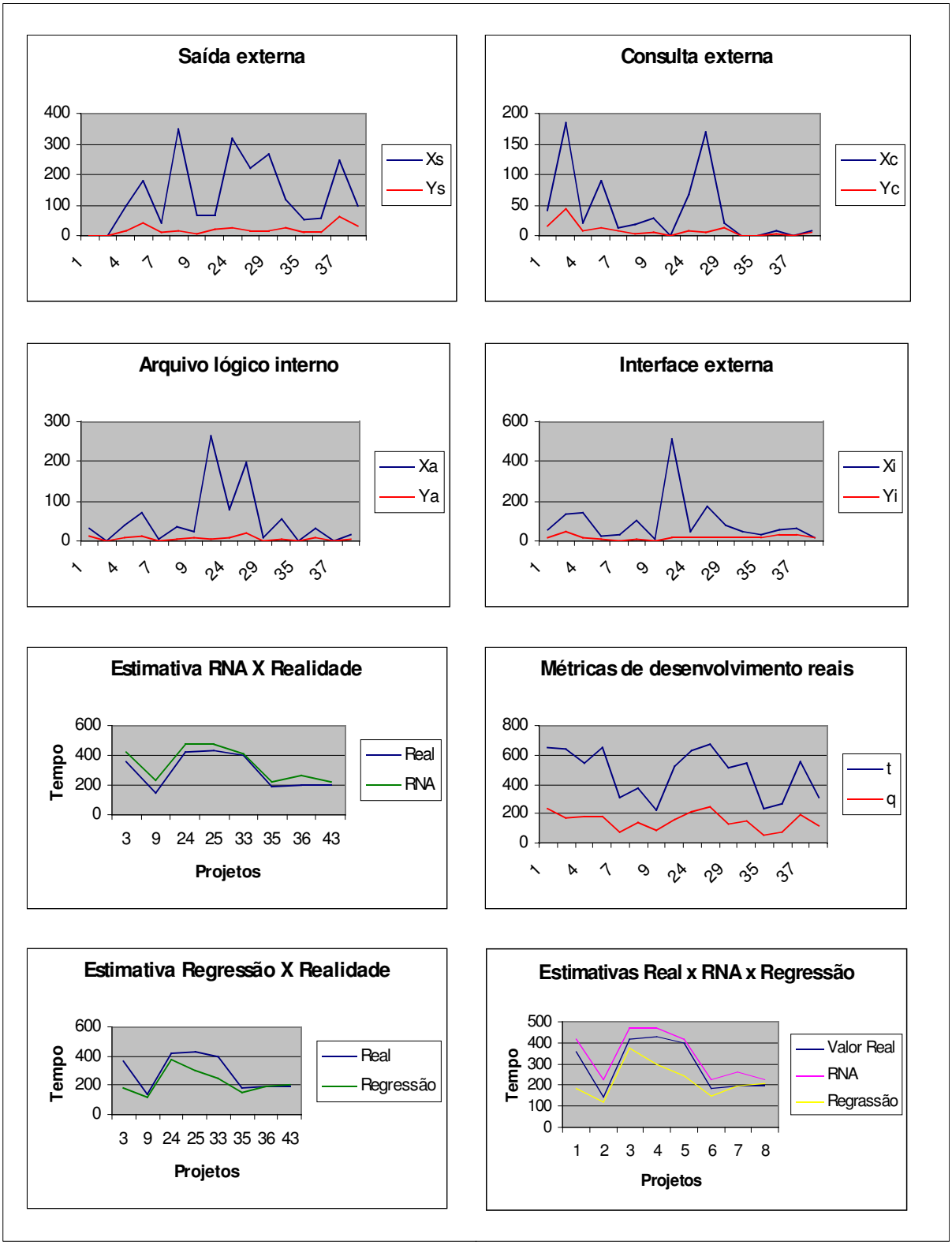

Figura 4.9 – Gráficos comparativos.

Excluindo-se os projetos acima citados, foram criados dois novos conjuntos de dados: um para treinamento e outro para testes, contendo, cada um, 50% dos projetos do

conjunto restante. Realizadas 50.000 interações, com tolerância máxima admitida de 1.0E-4, para treinamento, e fator de aprendizagem 0.8, obtiveram-se os seguintes valores de erro para a estimação do tempo de desenvolvimento por RNA/BP em 100 experimentos:

Medidas de dispersão da RNA/BP (***t***):

Erro médio percentual = -21,42;

Desvio padrão do erro = 17,55.

Usando o mesmo conjunto de variáveis de entrada, o modelo de Abran e Robillard (1996) com 10 variáveis fornece estimativas para o tempo de desenvolvimento com o desempenho abaixo mostrado:

Medidas de dispersão da Regressão (***t***):

Erro médio percentual = 20,31

Desvio padrão do erro = 18,59;

Conforme verificado, os valores fornecidos pelo modelo implementado são bem próximos dos obtidos pela utilização do modelo de regressão estatística, utilizado por Abran e Robillard (vide gráfico comparativo na Figura 4.9), mesmo sendo utilizados poucos dados no treinamento de RNA/BP. Tal resultado é um forte indício de que se pode relacionar diretamente os atributos básicos dos *softwares* com suas respectivas medidas de produtividade.

Utilizando-se uma quantidade maior de dados, gerados a partir da função de Boehm para um conjunto de *softwares* aleatoriamente criados, o item a seguir mostra uma avaliação de um protótipo de sistema estimador para métricas de desenvolvimento.

4.4.2 Protótipo de sistema estimador de tempo e esforço de desenvolvimento.

Uma rede neural artificial, com arquitetura auto-organizável (Apêndice A), é utilizada na implementação do módulo **Gerente** que foi responsabilizado em realizar um mapeamento topológico de possíveis classes de processo produtivo em uma camada de neurônios com dimensão inferior à de sua entrada. Para tanto, deve-se realizar o treinamento da rede utilizando-se padrões de dados relacionados a um conjunto de entidades caracterizadas pelos fatores [**f**, **m**, **p**, **e**, **o**] identificados no Capítulo 3.

Conforme mostra a Figura 4.10, além das cinco entradas identificadas, uma malha de pesos sinápticos compõe o mapa. Esta malha deve-se autoajustar para fornecer a identificação da classe de processo que melhor represente a entidade de desenvolvimento, baseando-se no algoritmo de Kohonen, onde apenas um dentre os neurônios da camada de saída é declarado vencedor. Tal neurônio seleciona o especialista responsável em realizar a estimação de tempo e de esforço de desenvolvimento na classe de processo produtivo identificada.

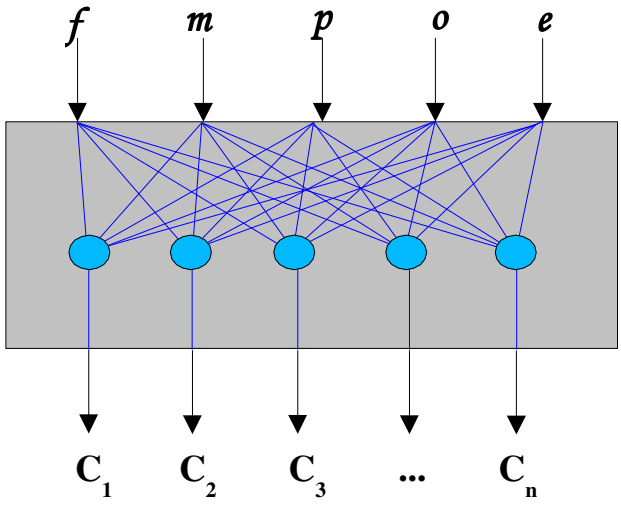

Figura 4.10 – Selecionador de redes homogêneas (mapa auto-organizável).

Determinadas as classes de desenvolvimento, deve-se utilizar bases de dados históricas para cada uma dessas classes para treinamento de especialistas estimadores. A Figura 4.11 mostra o modelo completo proposto neste trabalho, onde as redes $RN_q's$ (1 a n) serão redes especializadas em estimar o esforço e o tempo de desenvolvimento, como definido no Item 4.4.1.

A fase de treinamento das redes deve seguir os seguintes passos:

- criar a rede neural para o gerente (mapa de Kohonen) com uma pré-determinada quantidade de classes;
- com instâncias, aleatoriamente geradas de entidades, realizar seu treinamento;

- criar-se um especialista para ser treinado com cada banco de dados separado pelo gerente.

Com o gerente (mapa de Kohonen) e cada especialista treinados, o funcionamento do sistema estimador de tempo e de esforço de desenvolvimento é descrita da forma a seguir.

Primeiro, fornecem-se informações [f, m, p, o, e] sobre a entidade de desenvolvimento para o gerente, que indicará sua melhor classe representante de tal entidade. Segundo, o especialista treinado na classe selecionada recebe as informações caracterizadoras do *software* a ser desenvolvido (atributos básicos de AFP) e fornece estimativas a respeito do tempo e do esforço de desenvolvimento a ser realizado.

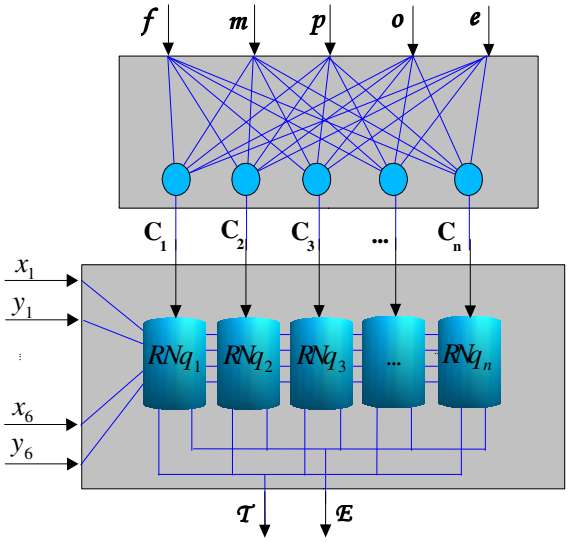

Figura 4.11 – Modelo completo para estimação de métricas.

A interface desse outro programa (protótipo) é mostrada na Figura 4.12. Como se pode ver, uma entidade de desenvolvimento *Ent-0* é composta por uma equipe *novaEquipe*, uma ferramenta *novaFerramenta*, uma metodologia *novaMetodologia*, uma plataforma *novaPlataforma* e uma organização *novaOrganização*. Cada um desses atributos é definido por multiplicadores de Boehm (como definido no Item 4.3.2).

O gerente mostrado foi criado com 100 neurônios em seu mapa neural. Devido à quantidade de entidades geradas aleatoriamente durante o treinamento, apenas 75 classes foram utilizadas.

Treinados os especialistas associados a cada classe, um teste com 100 entidades e 100 programas aleatoriamente gerados foi realizado, fornecendo os resultados mostrados na Figura 4.12. Para o tempo, o erro médio quadrático (EMQ) foi de 12.21% e de 15.64% para o esforço. Uma medida de desempenho das estimativas também foi fornecida. Para o tempo, o sistema alcança estimativas com erro inferior a 20% em 82.01% das vezes e em 75.42% das vezes para o esforço.

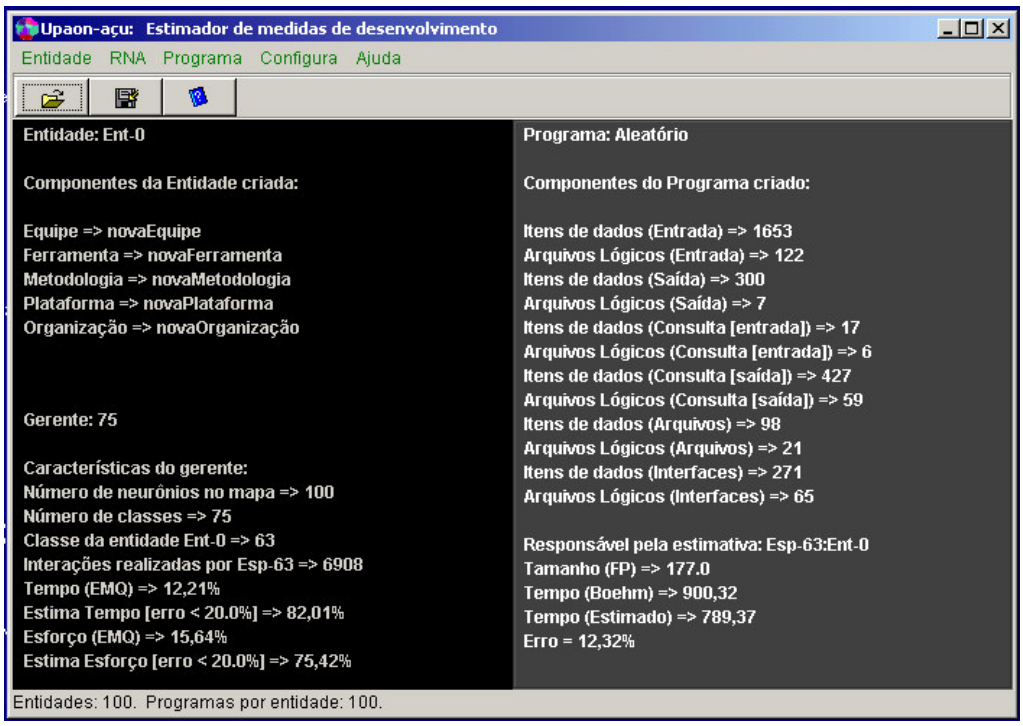

Figura 4.12 – Interface do protótipo implementado.

Também é mostrado, na interface em questão, informações sobre um programa submetido à avaliação quanto às métricas do tempo e do esforço de desenvolvimento. São mostradas as quantidades totais de cada atributo constituinte do referido *software*. Após classificada a entidade *Ent-0* no mapa criado (*Entidade|Classificar*), o especialista selecionado fornece estimativas a respeito do programa carregado. Os resultados fornecidos indicam que seu tamanho em pontos de função é de 177, que o tempo de desenvolvimento real (Boehm) é de 900.32 dias e que o tempo estimado seria 789.37 dias. O erro associado a tal estimativa é de 12.32%.

Deve ser mencionado que quando é fixada, durante o treinamento, a entidade produtora, isto é, considerando que não ocorram variações no processo de desenvolvimento

dos vários programas estimados, os resultados são ainda mais animadores. O EMQ do tempo seria de 5.84% e de 7.83% para o esforço. Para o tempo, as estimativas fornecem erros inferiores a 20% em 97.79% das vezes. Para o esforço, os erros fornecidos localizam-se abaixo de 20% em 94.25% das vezes.

Chulani et. alii. (1999) menciona em seu trabalho que na nova versão do COCOMO II (1998) apenas 63.60% das vezes a estimativa fornecida apresenta erro inferior a 20%. E que, mesmo após técnica estatística de estratificação por modelo *bayesiano* ser aplicada, o desempenho da estimativa é de 70% (Tabela 4.1).

COCOMO II	**Precisão da estimativa**	**Antes da estratificação**	**Após a estratificação**
1997	PRED(.20)	46.00%	49.00%
	PRED(.25)	49.00%	55.00%
	PRED(.25)	52.00%	64.00%
1998	PRED(.20)	63.60%	70.00%
	PRED(.25)	68.00%	76.00%
	PRED(.25)	75.00%	80.00%

Tabela 4.1 – Resultados apresentados por Chulani.

Não se pode apresentar uma afirmação contundente sobre os dados oferecidos pelo protótipo, já que os dados de treinamento não foram realmente executados, somente gerados a partir da função de esforço de Boehm, mas a comparação mostrada na Tabela 4.1 oferece uma confirmação para a tendência percebida na pesquisa e expressa pelo protótipo.

CAPÍTULO 5
CONCLUSÕES

A Engenharia de Software é responsável pelo planejamento, administração e controle do processo de desenvolvimento de seu respectivo produto: o *software*. Nesse contexto, as métricas de *software* representam um dos mais importantes fatores determinantes do sucesso de um bom projeto de desenvolvimento. A estimação de métricas de desenvolvimento é uma tarefa bastante complexa, devido a grande quantidade de variáveis que compõem sua modelagem. Tal dificuldade gerou a criação de muitos modelos e técnicas de estimação de medidas que buscam ser eficientes e simples.

Um conjunto de variáveis capazes de estimar métricas de produtividade como o tempo e o esforço de desenvolvimento foi levantado neste trabalho. A partir das visões de Pressman, Machado e Boehm, esse conjunto de variáveis foi organizado em dois módulos: um que identifica o processo de desenvolvimento e o outro que identifica o produto (*software*) desse desenvolvimento.

O conjunto de variáveis relativo ao processo de desenvolvimento foi agrupado em cinco fatores: ferramenta, metodologia, plataforma, equipe e organização. Essas variáveis seguiram o mesmo nível abstracional definido por Chiavenato (2001) para a Teoria Geral da Administração. Analogamente, a ferramenta representa a tecnologia utilizada pela entidade desenvolvedora, a metodologia relaciona-se às tarefas realizadas na construção dos *softwares*, a plataforma é semelhante ao ambiente no qual é realizado processo de construção, a equipe é formada pelas pessoas que trabalham na execução das tarefas construtivas e a organização adequa-se à estrutura em que se baseia a entidade para a sistematização do processo de desenvolvimento de *softwares*.

O conjunto de variáveis relativo ao produto foi definido a partir do modelo utilizado em FPA (IFPUG, 1994). Um problema de flexibilidade foi identificado nesse método, tanto na apuração dos pontos de função quanto no ajuste de complexidade, pois a granulosidade da métrica leva à redução no domínio de sua aplicabilidade. Desta maneira, considerou-se que os fatores relacionados ao produto, vistos a partir da visão do usuário,

poderiam ser definidos por seus atributos básicos, especificamente, os números de itens de dados, arquivos lógicos e registros lógicos de cada tipo de função componente do *software*.

Com o objetivo de facilitar o estudo, considerou-se também que com a utilização de diferentes fatores componentes do processo poder-se-ia agrupar as várias entidades desenvolvedoras em classes de processos que forneceriam desempenhos similares. A partir daí, o problema de estimação foi então dividido em duas partes: uma (gerente) seleciona a classe que melhor representa o processo de desenvolvimento a ser realizado; e a outra (especialista) que realiza propriamente as estimativas de tempo e esforço de desenvolvimento.

A parte gerente utilizou um sistema neural auto-organizável (Kohonen) capaz de realizar a separação dos diferentes processos produtivos em conjuntos específicos definidos por fatores de produtividade estabelecidos pelo modelo de Boehm.

Selecionado o processo produtivo específico de desenvolvimento, a parte especialista baseou-se em uma rede neural *feed forward* com *back propagation* para proceder a estimativas de métricas relativas ao *software* como o tempo e o esforço de desenvolvimento de *software*.

Baseado em um conjunto contendo 21 projetos disponibilizados por Abran e Robillard (1996), com fatores de produtividade homogêneos, separaram-se aqueles que possuíam menor desvio-padrão em relação à média, restando apenas 16 projetos, dos quais destinaram-se 8 (oito) para a realização de treinamento e 8 (oito) para testes da rede neural artificial utilizada para implementar o especialista estimador. A rede neural artificial forneceu resultados de estimativas do tempo de desenvolvimento de *softwares* cujas medidas de dispersão se colocaram muito próximas das fornecidas pelos modelos mostrados por Abran e Robillard, baseados em regressão estatística.

Assim, notou-se que, mesmo sendo parcos o dados utilizado no treinamento, a rede neural especialista "aprendeu" como se comportam as métricas de desenvolvimento em função seus correspondentes atributos funcionais primários, relativos a apenas um conjunto específico de características de produtividade. Esta talvez tenha sido a maior contribuição oferecida por esta dissertação, pois, de maneira geral, o mapeamento direto entre as métricas de tempo e de esforço e os atributos básicos dos *softwares* tornou menos subjetivo e mais preciso o processo de estimação, embora ainda apresente modelo matemático implícito (característica intrínseca da ferramenta de Inteligência Artificial utilizada).

O modelo apresentado neste trabalho foi especificado através de diagramas de casos de uso e de diagramas de classes definidos pela linguagem de projetos unificada (UML). Foram utilizados dois pacotes de *software*: o jahuwaldt.NeuralNets, que forneceu redes neurais para a implementação dos especialistas; e o jfröhlich.NeuralNets, que permitiu a implementação do gerente. Todo o projeto computacional mostrado foi implementado utilizando-se a linguagem multi-plataforma e orientada a objetos Java, baseada no ambiente de desenvolvimento JBuilder. Com isso, futuras adequações no protótipo desenvolvido serão simplificadas e agilizadas.

Da forma que foi implementado o modelo proposto neste trabalho, é possível o fornecimento de estimativas logo nas fases iniciais do ciclo de vida dos *softwares*. Classificado o processo existente em uma entidade desenvolvedora, o especialista fornecerá suas estimativas a partir apenas dos valores determinados pelos atributos básicos do *software* contratado pelo usuário. Em termos de simplicidade, o processo de estimação fica resumido à contagem desses atributos, tão logo ocorra a classificação do processo.

APÊNDICES

Apêndice A - Redes Neurais Artificiais.

Neste apêndice serão definidos os aspectos básicos das redes neurais artificiais. Duas arquiteturas serão frisadas: *feed forward* e mapa auto-auto-organizável (Kohonen).

A.1 Aspectos básicos das Redes Neurais

Rede neural artificial é um conjunto interconectado de unidades computacionais simples, chamados neurônios artificiais, processado paralelamente, cujo objetivo é realizar o mapeamento de padrões de entrada para a saída através de um prévio treinamento (Shaw e Simões, 1999).

Inicialmente, as conexões da rede, representadas por uma matriz de pesos sinápticos, fornecem um comportamento imprevisível, aleatório. Após a fase de treinamento, primeira parte do processo de mapeamento, a matriz modifica-se tal que consiga produzir as ações desejadas, passando assim para a segunda fase do processo: a estimação. Na fase de estimação é quando a rede reage aos estímulos de entrada conforme o desejado e nos limites de desempenho determinados por sua arquitetura, treinamento e qualidade de informações.

A disposição, em que o conjunto de neurônios artificiais, componentes da rede neural, é projetada, define a estrutura de funcionamento da rede, isto é sua arquitetura. Duas formas de arquitetura serão utilizadas nesta dissertação: a *feed forward* e o mapa de características de Kohonen.

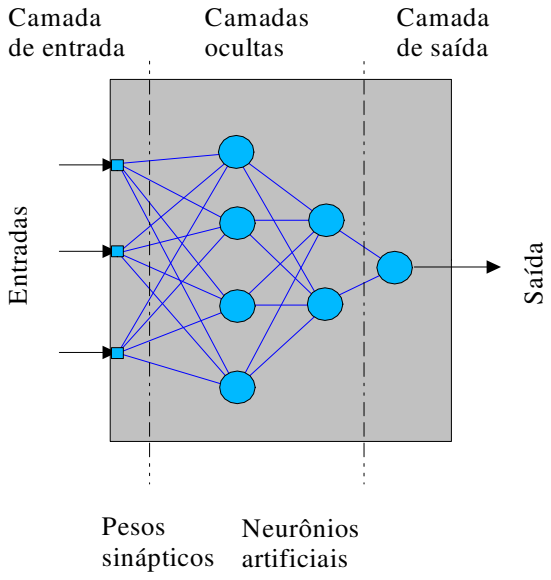

Figura A.1 – Rede neural artificial feed forward.

A.2 Arquitetura *feed forward*.

A arquitetura *feed forward* é aquela em que os neurônios da rede são dispostos em uma ou mais camadas que se projetam, total ou parcialmente, uma sobre a outra, unicamente, no sentido da entrada para a saída. Quando existem três ou mais camadas, a primeira é chamada de camada de entrada, a última de camada de saída e as centrais de camadas ocultas.

A Figura A.1 mostra uma rede neural artificial *feed forward* com quatro camadas: a camada de entrada, composta de três nós fontes; duas camadas ocultas, compostas de quatro e dois neurônios, respectivamente; e a camada de saída, com apenas um neurônio artificiais.

A.3 Mapa auto-organizável de Kohonen.

O mapa neural de características se trata de uma rede neural auto-organizável, desenvolvida por Kohonen, que possibilita a visualização de dados expressos na entrada em uma dimensionalidade maior que a da saída, através de um mapa uni ou bidimensional, com informações agrupadas segundo suas similaridades (Germano, 1999).

Haykin (2001) descreve um mapa auto-organizável como uma entidade que realiza a transformação de um padrão de entrada de dimensionalidade arbitrária em uma saída que se apresenta em uma ou duas dimensões topológica e adaptativamente ordenada. Ele afirma que tal mapa é composto de três subprocessos: **competição**, **cooperação** e **adaptação sináptica**.

No processo de competição os neurônios da saída calculam seus valores para uma função discriminante, assinalando aquele que obteve o maior valor como vencedor. Esse neurônio é localizado no centro de uma vizinhança topológica, onde, cooperativamente, influencia sobre a excitação daqueles neurônios que estiverem mais próximos. Finalmente, o processo de adaptação sináptica é realizado através da inclusão de um termo de *esquecimento* à hipótese hebbiana, isto é, em vez de se modificarem as conexões em uma só direção, o que levaria a rede à saturação, o termo de esquecimento gerará um movimento d vetor peso sináptico do neurônio vencedor em direção ao vetor de entrada, o que leva à ordenação topológica do mapa de características em discussão.

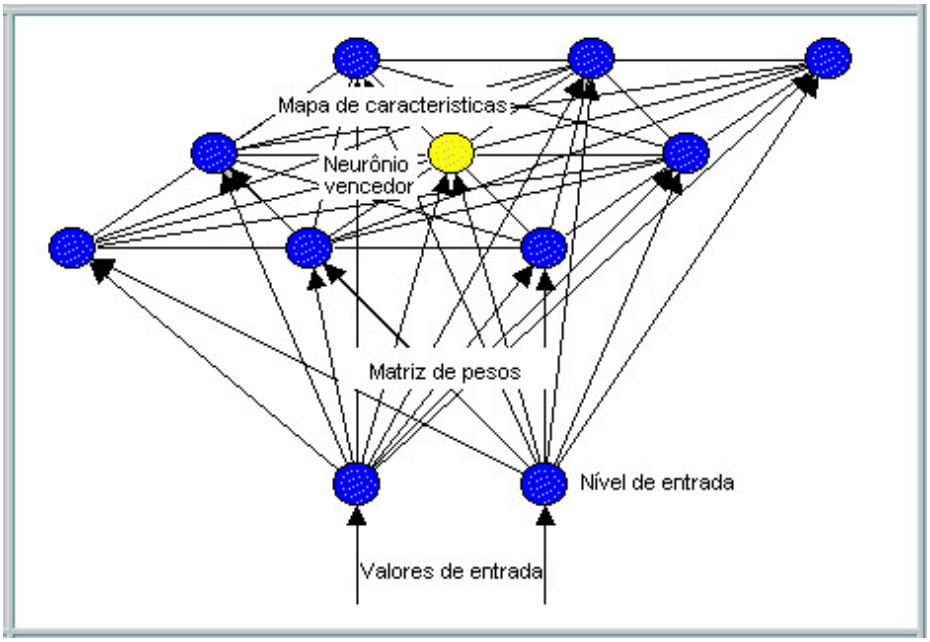

Figura A.2 – Mapa auto-organizável (Kohonen).

A arquitetura criada por Kohonen é uma das mais comercialmente utilizadas, sendo capaz de realizar a compressão de dados e de mapear uma quantidade pré-especificada de valores de entrada, de maneira topologicamente otimizada. O algoritmo utilizado por Kohonen pertence à classe dos algoritmos de codificação vetorial. Suas principais características, segundo JOPPE et. alli (1990), são:

- Possui auto-aprendizado, entrada intervalar e conexão competitiva.
- Não há propriamente um reconhecimento de padrão como em outros modelos, mas há a classificação de um padrão junto com outros que têm características semelhantes, formando classes. Estas classes são organizadas num mapa, onde se pode observar a distribuição dos padrões. Desta maneira, no instante em que um padrão é inserido na rede, esta o coloca na classe onde melhor o padrão se adequa, em função das suas características.
- Um outro aspecto importante é que o modelo de Kohonen é chamado de biologicamente plausível. No córtex auditivo, por exemplo, existem conjuntos de células que só reagem a determinados impulsos ou freqüências, enquanto a outros não. No modelo ocorre o mesmo, onde um padrão ao ser reconhecido faz com que um ou somente alguns neurônios de saída sejam ativados (aqueles que mais se assemelham ao padrão inserido) enquanto outros não.
- Este tipo de rede é usado quando se deseja, por exemplo, reconhecer diversos padrões que possuam alguma relação entre si, como reconhecimento de voz, que será explicado posteriormente na seção de aplicações.

A.4 Formas de ajuste de estrutura: treinamento e aprendizado.

Para o fornecimento do comportamento esperado, tal que possa ser utilizada de forma útil, a rede neural passa pela fase de treinamento, em que ela "aprende" a relacionar dois conjuntos de informações distintos (entrada e saída). O aprendizado utilizado nesta fase pode ser definido como um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede esta inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre (Haykin, 2001).

Quando a aprendizagem utiliza um professor, que fornece a ação ótima a ser realizada pela rede, ela é dita supervisionada. Neste tipo de aprendizado, o conhecimento é representado por um conjunto de padrões entrada-saída, onde a rede tenta fazer com que a diferença entre sua estimativa e sua saída (erro) esteja dentro de um patamar predefinido.

O outro tipo de treinamento é aquele em que os parâmetros livres da rede são ajustados através de um algoritmo de aprendizagem não-supervisionada. Neste tipo de aprendizagem, não há a necessidade de um professor. É sim utilizada a competição entre os neurônios de saída, buscando a melhor representação do padrão na entrada (Rich e Knight, 1993).

A.5 Arquitetura *feed forward* com aprendizado *back propagation*.

Back propagation é a técnica específica de implementação do algoritmo de aprendizado supervisionado mais utilizada. Ela se baseia na regra de aprendizagem por correção de erro, uma generalização do algoritmo do mínimo quadrado médio. Este algoritmo consiste de uma fase de propagação, em que um padrão é aplicado à entrada e transmitido através de todos os pesos sinápticos (fixos) da rede até a sua saída, e em uma fase de retropropagação, em que a diferença entre a saída produzida e a saída desejada é propagada no sentido inverso, tal que os pesos sinápticos sejam alterados em toda a rede, proporcionalmente a este, tornando o comportamento da rede mais próximo do almejado.

Normalmente o processo de aprendizagem se dá através de um grande número de repetições, passos ou interações, isto é, um grande número de aplicações do algoritmo aos padrões entrada-saída apresentados para treinamento. Ele prossegue até que seja atingido um valor para o erro de saída da rede inferior ao estipulado. Ele pode também ser forçado a

reduzir um parâmetro qualitativo, a *máxima tolerância admitida*, que é definida como o maior valor permitido para o erro gerado por cada neurônio artificial da rede.

Outro parâmetro de qualidade do aprendizado da rede é o *fator de aprendizagem*. Ele é definido por Haykin (2001) como a relação entre dois valores. A variação absoluta dos valores de cada um dos pesos sinápticos e respectiva taxa de variação de uma função custo em relação a cada peso sináptico conectado a um neurônio da rede, em determinada iteração. A função custo mencionada refere-se à soma instantânea dos erros médios quadráticos gerados pelos neurônios.

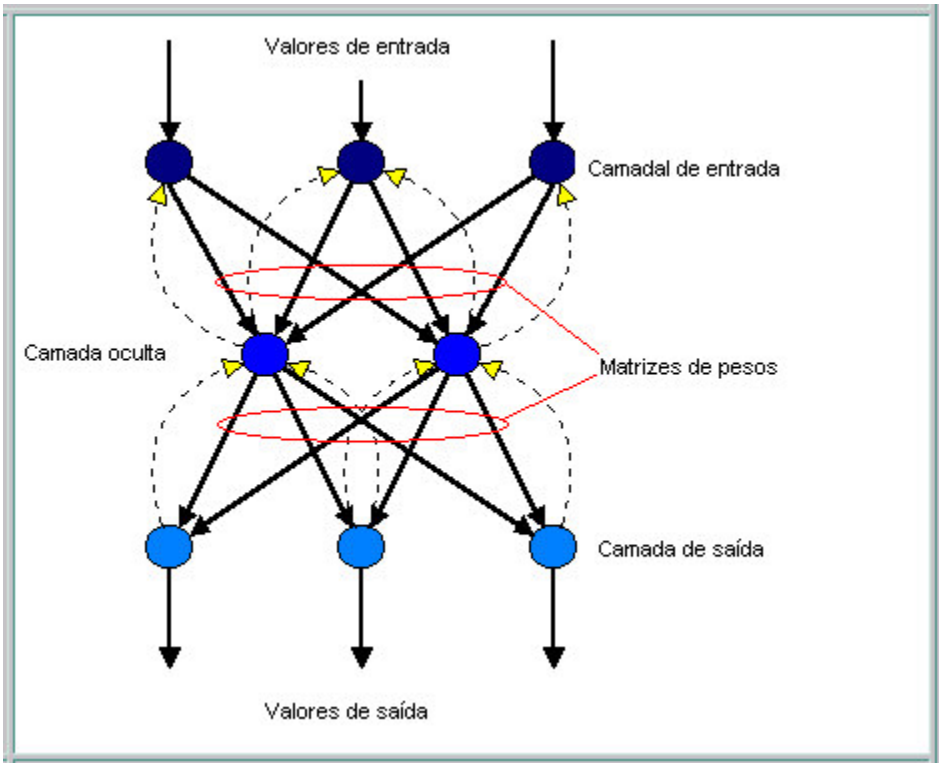

Figura A.3 – Arquitetura feed forward.

Rich e Knight (1993) definem o **fator de aprendizagem** como *um fator de escala que informa a distância que devemos mover-nos na direção do gradiente*, onde este gradiente é um vetor que indica a direção da redução do erro médio quadrático supracitado. Ou seja, se ao fator de aprendizagem for atribuído uma valor pequeno, o aprendizado pode se tornar muito lento. De outro modo, se tal fator possuir valor muito alto, o aprendizado da rede pode ser prejudicado, fazendo com que o ponto ótimo seja ultrapassado e levando-a à instabilidade.

Devido a sua estrutura e funcionamento, as redes neurais artificiais disponibilizam funcionalidades como a tolerância à falta de informações na entrada, potencial de generalização e poder de aprendizagem e de estimação. Todas estas funcionalidades advém de sua capacidade de mapeamento de dados históricos de entrada e saída.

Mas, as redes neurais oferecem dificuldades como incapacidade de descrição da solução obtida, não esclarecem as possíveis causas dos fenômenos modelados, são projetadas por regras *ah-doc* e são limitadas a modelamento de fenômenos de dinâmica lenta (SHAW e SIMÕES, 1999).

Apêndice B - Projetos utilizados por Abran e Robillard (1996)

ID	X1	Y1	X2	Y2	X3	Y3	X4	Y4	X5	Y5	X6	Y6	FP	Dias
0	251	56	0	0	6	4	34	11	32	12	52	17	233	418
1	0	0	0	0	13	13	172	30	0	0	138	46	167	468
2	42	21	99	17	10	2	11	6	40	6	143	16	179	360
3	105	24	133	17	24	9	119	16	57	11	111	27	255	531
4	84	15	178	42	25	5	66	9	70	11	24	9	181	471
5	784	51	204	16	0	0	0	0	92	12	12	1	221	525
6	29	15	40	12	3	3	11	5	4	1	33	2	80	225
7	57	8	348	17	1	1	18	2	34	4	103	8	144	229
8	44	12	68	7	2	2	25	4	22	6	5	2	83	143
9	90	11	66	22	0	0	0	0	265	5	517	18	157	369
10	78	17	320	24	2	1	64	7	79	8	49	14	216	416
11	215	28	219	17	8	2	160	4	199	20	176	16	242	428
12	7	7	266	16	4	3	17	11	6	1	78	13	132	377
13	13	7	170	100	31	14	10	5	11	10	87	55	258	544
14	3	3	33	5	0	0	3	2	1	1	3	2	39	52
15	112	8	116	25	0	0	0	0	57	5	44	17	145	400
16	17	1	50	9	0	0	0	0	0	0	28	15	51	187
17	41	1	58	10	3	1	4	2	33	7	59	31	72	198
18	2	2	246	64	0	0	0	0	0	0	65	28	194	363
19	12	2	98	29	2	2	5	2	15	4	18	12	116	195
20	24	4	26	5	2	2	13	4	47	7	21	4	69	69

Apêndice C - Pacote jfröhlich.NeuralNets

Este item descreve as principais características do pacote criado por Fröhlich (1996).

Passos para criação de um mapa de características de Kohonen		
passo	código simples	descrição
1. declaração do objeto	KohonenFeatureMap kfm;	EXIGIDO Declara o objeto kfm como sendo do tipo KohonenFeatureMap.
2. inicialização do objeto	kfm = new KohonenFeatureMap();	EXIGIDO Cria uma instância da classes KohonenFeatureMap.
3. cria um mapa de características	kfm.createMapLayer(xSize,ySize);	EXIGIDO Cria uma mapa de características com xSize*ySize neurônios.
4. define matriz a matriz de entrada	Descrito na tabela abaixo	EXIGIDO Cria uma matriz de entrada.
5. conecta o nível de entrada com o mapa de características	kfm.connectLayers(im);	EXIGIDO Conecta automaticamente cada neurônio de entrada com cada neurônio do mapa de características. Todos os neurônios do mapa são também conectados entre si. Todos os pesos das matrizes são criados e inicializados aleatoriamente, baseando-se na matriz de entrada im.
ajusta taxa de aprendizagem	kfm.setInitLearningRate(x);	OPCIONAL Ajusta a taxa de aprendizagem inicial em x. Seu valor padrão é 0,6.
Ajusta área de ativação inicial	kfm.setInitActivationArea(x);	OPCIONAL Ajusta área inicial de ativação em x, Seu valor padrão é dado pelo tamanho do mapa dividido por dois.
Ajusta a área final de ativação	kfm.setStopArea(x);	OPCIONAL Ajusta a área final de ativação em x. Seu valor padrão é um décimo da área de ativação inicial.

Ajusta número máximo de ciclos de aprendizagem	kfm.setMaxLearningCycles(x);	OPCIONAL Ajusta número máximo de ciclos de aprendizagem em x. O seu valor *default* é -1.
Reinicia contagem de tempo de aprendizagem	kfm.resetTime();	OPCIONAL Reinicia contagem de tempo de aprendizagem.
6. Inicia ciclos de aprendizagem	kfm.learn();	EXIGIDO Este método é geralmente usado dentro de um *loop*, que pára quando a condição finishedLearning() se torna falsa.

Métodos que disponibilizam de informações		
Área de ativação	kfm.getActivationArea();	OPCIONAL Retorna a área corrente de ativação no mapa de características.
Tempo decorrido	kfm.getElapsedTime();	OPCIONAL Retorna o tempo de aprendizagem decorrido.
Área final de ativação	kfm.getStopArea();	OPCIONAL Retorna a área final de ativação no mapa de características.
Área inicial de ativação	kfm.getInitActivationArea();	OPCIONAL Retorna a área inicial de ativação no mapa de características.
Taxa inicial de aprendizagem	kfm.getInitLearningRate();	OPCIONAL Retorna a taxa inicial de aprendizagem da rede.
Ciclo de aprendizagem	kfm.getLearningCycle();	OPCIONAL Retorna o ciclo de aprendizagem atual da rede.
Taxa de aprendizagem	kfm.getLearningRate();	OPCIONAL Retorna a taxa de aprendizagem corrente da rede.
Tamanho do mapa na direção x	kfm.getMapSizeX();	OPCIONAL Retorna o tamanho do mapa na direção da dimensão x.
Tamanho do mapa na direção x	kfm.getMapSizeY();	OPCIONAL Retorna o tamanho do mapa na direção da dimensão y.

Número de pesos	kfm.getNumberOfWeights();	OPCIONAL Retorna o número de pesos da matriz que conecta as entradas com os neurônios do mapa de características.
Valores dos pesos	kfm.getWeightValues();	OPCIONAL Retorna os valores dos pesos da matriz que conecta as entradas com os neurônios do mapa de características.

Passos para utilização de Input Matrix		
passo	código simples	descrição
1. declaração de objeto	InputMatrix im;	EXIGIDO Declara um objeto do tipo InputMatrix.
2. chamada do construtor	im = new InputMatrix(size,dim);	EXIGIDO Cria uma instância da classe InputMatrix. Seu construtor recebe os seguintes argumentos: size – número de entradas (1...N); dim - dimensão da matriz (1...3)

3. Ajusta quantidades de neurônios em cada direção	Individualmente:
	im.setInputX(x);
	im.setInputY(y);
	im.setInputZ(z).
	
	Todas de uma só vez:
	im.setInputValues(x,y,z).
		EXIGIDO
		Ajusta as quantidades de neurônios em cada dimensão da matriz. Esses métodos podem ser utilizados separada ou atomicamente.
		Os argumentos x, y, e z são vetores de inteiros.

Métodos que disponibilizam de informações		
Número de valores de entrada	im.size();	OPCIONAL Retorna o número de valores de entrada da matriz im.
dimensão	im.getDimension();	OPCIONAL Retorna a dimensão da matriz de entrada im.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAN, Alain e ROBILLARD, F. **Function Points Analysis: an empirical study of its measurement processes**. IEEE, v. 22, n. 12, dec. 1996.

ABRAN, Alain. **Function Points Analysis: a study of their measurement processes and scale transformations**. Québec : EPM. 1994.

BADE, R. et. alii. **A systems engineering capability maturity model**. Version 1.1. Software Engineering Institute. CMU/SEI-95-MM-003, Nov./1995.

BOEHM, B. W. et. alii. **Cost models for future software life cycle processes: COCOMO 2.0**. Amsterdam, 1998.

BOOSH, G. **Object-Oriented Analysis and Design with Applications**. The Benjamin/Cummings Publishing Company Inc., Redwood City, California, 1994.

BÜREN, G. e KOLL, I. **First experiences with the introduction of an effort estimation process.** Paris : ICSSEA 99, 1999.

CHIAVENATO, Idalberto. **Teoria geral da administração**. Rio de Janeiro : Campos, 2001.

CHULANI, S., et. alii.**Bayesian Analysis of Empirical Software Engineering Cost Models**. IEEE Transactions on Software Engineering, Vol. 25, Nº. 4, July/August 1999.

CHULANI, S. et. alii. **Software development cost estimation approaches - a survey**. Los Angeles : University of Sourthern California, 1998.

CNI/COMPI, **Metrologia: conhecendo e aplicando na sua empresa**. Brasília : Confederação Nacional da Indústria, 2000.

DEKKERS C. A. **Desmistificando pontos de função: entendendo a terminologia**. Semilnole : Quality Plus Tecnologies, 1998.

DESHARNAIS, J-M. e ABRAN, A. **How to successfully implement a measurement program: from theory to practice**. Québec : UQAM, 1999.

DIVERIO, T. A. e MENEZES, P. B. **Teoria da Computação: máquinas universais e computabilidade**. Porto Alegre : Instituto de Informática da UFRGS : Editora Sagra Luzzatto, 1999.

FENTON, N. E., PFLEEGER, S. L., **Software metrics - a rigorous and practical approach**. International Thomson Press, Boston, 1997.

FORRESTER, J. W. **Industrial Dynamics**. Cambridge, MIT Press, 1961.

FRÖHLICH, J. **Pacote jfröhlich.NeuralNets**. Copyright © 1996-97. URL: http://rfhs8012.fh-regensburg.de/~saj39122/jfroehl/diplom/e-index.html, 1996.

GERMANO, T. **Self organizing maps**. URL: http://davis.wpi.edu/~matt/courses/soms/, 1999.

GOODBRAND, A. **Software size estimation**. Calgary : University of Calgary, 1997.

HAYKIN, S. **Redes neurais: princípios e prática**. Porto Alegre : Bookman, 2001.

HUWALDT, J. **Pacote jahuwaldt.NeuralNets**. Copyright © 1999. URL: http://seagis.sourceforge.net/javadoc/seas/net/seas/neural/package-summary.html, 1999.

IFPUG. **Function Point Counting Practices Manual Relise 4.0**. Janeiro, 1994.

JOHNSON, K. **Software size estimation**. Calgary : CPSC, 1998.

JONES, C. e JONES, T. C. **Applied software measurement: assuring productivity and quality**. McGraw Hill, 1991.

JOPPE, A. et. alii. **A Neural Network for Solving the Traveling Salesman Problem**. Proceedings IEEE International Joint Conference on Neural Networks 3, 961-964, 1990.

KOKOL, P. **A wishful complexity metric**. Maribor : FERI, 2000.

LEITE, J. **Notas de aula de Engenharia de Software**. URL: http://www.dimap.ufrn.br /~jair /ES / indice.html. Natal : UFRN, 2000.

LOKAN, C. J. e ABRAN, A. **Multiple viewpoints in functional size measurement**. Lac Supérieur : IWSM'99, 1999.

LOKAN, C. J. **An empirical analysis of function point adjustment factors**. Camberra : University of New South Wales, 1999.

LONGSTREET, D. **Fundamentals of function point analysis**. URL: http:// www. softwaremetrics.com, 2000.

MACHADO, J. A. **A medição da produtividade no desenvolvimento de software**. Tese de mestrado da Universidade Federal do Rio de Janeiro. 1989.

OLLIGNY, S. et. alii. **A method for measuring the functional size of embedded software**. Québec : UQAM, 1999.

PETERS, J. F., PEDRYCZ, W. **Software engineering: an engineering approuch**. Washington : Wiley, 1999.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo : MAKRON Books, 1995.

PRICE, T. **Programa de Especialização: opção para o Mestrado**. Porto Alegre : UFRGS, 1997.

RICH, E., KNIGHT, K. **Inteligência artificial**. São Paulo : MAKRON Books, 1993.

SELNER, C. **Análise de requisitos para sistemas de informações, utilizando as**

ferramentas da qualidade e processos de software. URL: http://www.eps.ufsc.br/disserta99/selner. Santa Catarina : UFSC, 1999.

SHAW, I. e SIMÕES, M. **Controle e modelagem fuzzy**. São Paulo : Editora Edgard Blücher, 1999.

SLACK, N. et. alii. **Administração da produção**. São Paulo : Atlas, 1999.

ST-PIERRE, D. et. alii, **Adapting function points to real-time software**. Québec : UQAM, IFPUG Fall Conference, 1997.

SUNAT. **Roteiro oficial de práticas de contagem de pontos de função na SUNAT**. Rio de Janeiro : SERPRO/SUNAT, ver. 1.2, 2000.

SYMONS, C. **Function Point Analysis: difficulties and improvements**. IEEE Transactions on *software* Engineering, 1-14, Jan./1988.

VANDOREN, E. **Function point analysis: software technology review**. URL: http://www.sei.cmu.edu/str/descriptions/fpa_body.html : Carnegie Mellon University, Fev. 2001.

WEBER, K. et. alii. **Qualidade e produtividade em software**. São Paulo : Makron Books, 2001.

WU, L. **The comparison of the software cost estimating methods**. Calgary : CPSC, 1997.

YOURDON, E. **Análise estruturada moderna**. 10ª ed. Rio de Janeiro : Campus, 1990.

ZUSE, H. **A framework of software measurement**. Berlin : de Gruyter, 1998.