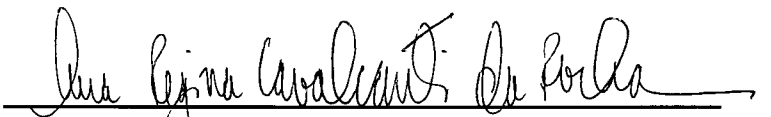
UMA FERRAMENTA PARA SELEÇÃO DE MÓDULOS REUTILIZÁVEIS

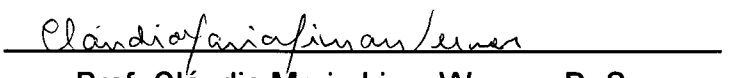
Cesar Augusto Comerlato

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

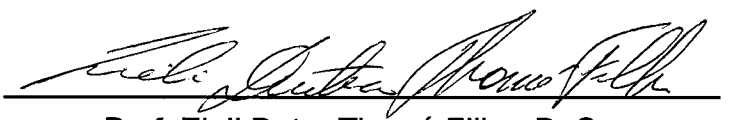
Aprovada por:



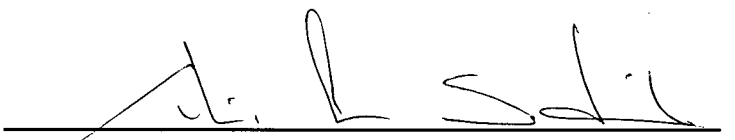
Prof. Ana Regina Cavalcanti da Rocha, D. Sc.
(Presidente)



Prof. Cláudia Maria Lima Werner, D. Sc.



Prof. Zieli Dutra Thomé Filho, D. Sc.



Prof. Júlio Cesar Sampaio do Prado Leite, Ph. D.

RIO DE JANEIRO, RJ - BRASIL
ABRIL DE 1994

COMERLATO, CESAR AUGUSTO

Uma Ferramenta para Seleção de Módulos Reutilizáveis [*Rio de Janeiro*] 1994,

viii, 112 p. 29,7 cm (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1994)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Reutilização de Software

I. COPPE/UFRJ II. Título (série)

À minha querida irmã Nani.

AGRADECIMENTOS

À minha orientadora Ana Regina pela paciência, incentivo e orientação ao longo deste trabalho.

Ao Aquilino, Schirru e todo o pessoal do Laboratório de Monitoração de Processos do Programa de Engenharia Nuclear, pelo apoio, incentivo e compreensão para a realização deste trabalho, sem os quais teria sido praticamente impossível.

Aos professores Júlio Cesar, Zieli e Cláudia, pela participação na banca examinadora.

Aos colegas Geraldo Xexéo e Eduardo Faria pela sua disponibilidade em discutir idéias e trocar experiências.

À Cris pela ajuda na editoração desta tese.

A todos da secretaria da Coppe/Sistemas que atenderam a todas as minhas solicitações com presteza e eficiência.

Aos meus amigos Célia e José Luiz pela sua disponibilidade em ajudar nas mais diversas situações e necessidades.

Aos meus irmãos Karla e Gian Carlo pela paciência e apoio ao longo deste trabalho, e também à minha irmã Karina pelo empréstimo do computador sem o qual teria sido difícil concluir este trabalho.

Aos meus pais e irmãos pela minha educação, amor e carinho.

A todos, que de uma ou outra forma, me auxiliaram na conclusão deste trabalho.

A Deus, Criador de tudo, pela minha energia, disposição e saúde.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M. Sc.).

UMA FERRAMENTA PARA SELEÇÃO DE MÓDULOS REUTILIZÁVEIS

Cesar Augusto Comerlato
ABRIL, 1994

Orientador: Prof. Ana Regina Cavalcanti da Rocha
Programa: Engenharia de Sistemas e Computação

Esta tese apresenta uma ferramenta para identificação e seleção de componentes candidatos à reutilização, a partir de sistemas já existentes, escritos em FORTRAN.

A ferramenta permite identificar e selecionar componentes de código candidatos à reutilização através da avaliação de variáveis linguísticas e lógica nebulosa (*fuzzy*). As variáveis linguísticas representam atributos de qualidade, definidos a partir da análise estática do código fonte com o uso de métricas de software. Esta tese apresenta, também, uma visão geral da reutilização de software e as principais questões relacionadas à reutilização.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M. Sc.).

A TOOL FOR REUSABLE MODULES SELECTION

Cesar Augusto Comerlato
APRIL, 1994

Thesis Supervisor: Prof. Ana Regina Cavalcanti da Rocha
Department: Systems and Computation Engineering

This thesis presents a tool to identify and select reusable candidate components from already existing software systems written in FORTRAN.

The tool allows the identification and selection of code components reusable candidates using linguistic variables and fuzzy logic evaluation. The linguistic variables represent quality attributes, defined from static analysis of the source code using software metrics. This thesis also presents an overview of software reuse and the main issues related to reuse.

Sumário

1 INTRODUÇÃO

1.1 Ciência da Computação e Reutilização de Software

Na história da humanidade, o final do século vinte será lembrado pelo advento do computador e sua participação na sociedade. A evolução dos computadores ao longo da segunda metade do século vinte é extraordinária.

A evolução da tecnologia de integração de componentes eletrônicos acelerou o desenvolvimento dos computadores. Eles passaram então a apresentar maior capacidade de memória e armazenamento, maior capacidade e velocidade de processamento e maiores índices de integração. Ao mesmo tempo a industrialização e difusão dessas novas tecnologias, permitiu uma considerável queda nos custos, resultando assim numa diminuição dos preços dos computadores e periféricos. Assim, computadores passaram a ser usados na resolução de problemas cada vez maiores e mais complexos, abrangendo cada vez mais, novas áreas de aplicação.

Entretanto, o ritmo de evolução da tecnologia de desenvolvimento de software não acompanhou a do hardware. A crise do software foi detectada em meados da década de sessenta, caracterizando-se pelo valor excessivo dos custos do software com relação aos custos do hardware. Esta defasagem ocorreu devido ao ponto crítico alcançado na época, de uma falta quase que total de metodologias e técnicas para o desenvolvimento do software.

Foi neste contexto que surgiu a Engenharia de Software, cujo objetivo era resolver as questões relacionadas ao desenvolvimento de software. Com o passar do tempo esta disciplina criou novos conceitos, paradigmas e áreas relacionadas ao desenvolvimento e manutenção do software.

Apesar do advento da Engenharia de Software e o aparecimento de novas técnicas, modelos, metodologias, ferramentas, ambientes integrados e novas linguagens para o desenvolvimento de software, a defasagem entre a evolução do hardware e do software ainda é notável. A relação custo/desempenho do hardware vem diminuindo 20% por ano, enquanto que a produtividade no processo de criação de software tem aumentado somente de 3% a 8% por ano, nos últimos anos [BIGG89a].

Atualmente o preço dos computadores permite que eles sejam empregados nos mais variados tipos de aplicação. Esta vasta abrangência computacional implica num aumento da demanda de software e, conseqüentemente, requer um maior número de profissionais devidamente qualificados, para atender a estas novas necessidades.

Produtividade e qualidade continuam sendo aspectos críticos no desenvolvimento de software. Os projetos de software são cada vez mais solicitados a fazer mais com menos recursos: entregar os sistemas solicitados em prazos menores, reduzir custos e tempo de manutenção, aumentar os níveis de desempenho e confiabilidade, aumentar a segurança dos sistemas, etc [BERS91], [BASI92]. Neste contexto, são imprescindíveis mudanças significativas na forma como o software é produzido atualmente.

Uma abordagem do problema de aumento de qualidade e produtividade pode ser resumida em três pontos principais: (1) otimizar a eficiência do processo; (2) reduzir a quantidade de trabalho refeito; (3) reutilizar produtos do ciclo de vida [BASI92].

Foi Dough MacIlroy, em 1968, quem pela primeira vez propôs o conceito de reutilização de partes de código fonte na indústria de software [CONT86]. O conceito de reutilização de software surge da constatação de que a cada novo projeto de sistema, os projetistas partem praticamente do zero, contando somente com as suas respectivas experiências na área. A "reinvenção da roda" a cada novo sistema implica, muitas vezes, em refazer produtos ou partes de produtos que já existem e, na maioria dos casos, nada do que já existe é aproveitado.

Alguns autores constataram a repetição de desenvolvimento do mesmo código em aplicações da mesma área, alcançando um índice de 60% de código comum [LANE89]. Ainda segundo Paulsein [CONT86], 50 a 95% da atividade dos programadores consiste de alteração de programas existentes. Conforme relatório do IBM Santa Teresa Laboratory, 77% do código escrito tem por objetivo adicionar novas características a produtos existentes [CONT86].

A reutilização de componentes de software existentes em novos sistemas implica numa menor produção de software novo, causando um aumento da produtividade final, qualidade e confiabilidade, mantendo a funcionalidade almejada. O aumento da produtividade é devido à diminuição do esforço necessário para a produção de código novo. O aumento da qualidade advém do fato do código reutilizado já ter sido amplamente usado, modificado e testado em outros sistemas. Estes mesmos motivos justificam o aumento da confiabilidade.

O conceito de reutilização, desde que foi proposto por MacIlroy, vem mudando com o passar do tempo. Tornou-se mais abrangente, deixando de estar vinculado somente à reutilização de código e passando a abranger um conceito de reutilização de qualquer item associado ao projeto de software, incluindo o conhecimento. Portanto, a reutilização envolve muito mais do que simplesmente código, pois envolve organização e encapsulamento de experiência e o estabelecimento de mecanismos e estruturas organizacionais para suportar este processo [BERS91].

As várias tecnologias de reutilização devem ser avaliadas, analisadas e testadas, para que se possa alcançar um nível máximo de reutilização em todas as etapas do ciclo de vida do software. As linguagens orientadas a objetos são um bom exemplo de aplicação prática da reutilização de software [MEYE87]. A maioria das linguagens de quarta geração (4GL), também, já utiliza algum tipo de reutilização de software.

É indispensável uma tecnologia que permita o "armazenamento" do conhecimento adquirido de forma que ele possa ser recuperado e reutilizado em novos projetos, pois este conhecimento e experiência adquiridos são fundamentais e não podem ser perdidos ou desconsiderados. Portanto, para que a reutilização de software se concretize, é necessária a definição de uma

estratégia que represente o melhor ou melhores caminhos para a sua real concretização.

1.2 A Linguagem FORTRAN

Já se passaram quarenta anos desde que foi aceita a idéia de John Backus para a primeira linguagem de programação de alto nível chamada FORTRAN (**FOR**mula **TRAN**slation). Era dezembro de 1953, e antes dessa data, ninguém concebia comunicar-se com um computador num nível superior ao da linguagem assembler [APPL91]. Mas a linguagem FORTRAN, além de revolucionária, foi inovadora. Ela tornou os computadores acessíveis a qualquer cientista ou engenheiro disposto a gastar um pouco do seu tempo no aprendizado da linguagem; não era mais preciso ser um especialista em computação para se desenvolver programas aplicativos [METC91].

A sobrevivência da linguagem FORTRAN está diretamente relacionada a padrões, que por sua vez estão intimamente ligados à aceitação e políticas. Devido ao tanto que já foi investido em código FORTRAN, escrito e otimizado, ele continuará a ser utilizado por muito tempo. FORTRAN é a linguagem preferida da comunidade científica nacional e internacional para o desenvolvimento de software nas áreas científica e de engenharia [APPL91].

Outros aspectos que justificam a "imortalidade" do FORTRAN são apresentados por Chivers e Clark [CHIV85]:

- o acervo de software já produzido é muito grande e a conversão deste software para outra linguagem seria uma tarefa extremamente custosa;

- a existência de um padrão largamente aceito e obedecido pelos compiladores disponíveis no mercado;

- a implementação da linguagem é muito eficiente gerando código de alto desempenho;

- os elementos da linguagem são de fácil assimilação.

Acrescentam ainda que os usuários da linguagem (em sua maioria pesquisadores, cientistas e engenheiros), por estarem mais preocupados com a solução de problemas específicos de suas áreas, encaram o desenvolvimento de software como uma atividade secundária [CHIV85].

Talvez seja este um dos principais motivos da sobrevivência da linguagem FORTRAN neste meio. Se o principal grupo de usuários da linguagem considera a atividade de desenvolvimento de software como atividade secundária, a conversão do software existente, ou a migração, para uma nova linguagem não faz nenhum sentido sob esta ótica.

Ainda sob esta visão, Cross [CROS86] observa que os responsáveis técnicos pelo desenvolvimento e manutenção de software científico consideram-se mais como cientistas, engenheiros ou matemáticos do que como engenheiros de software ou analistas de sistemas, apesar de gastarem a maior parte do seu tempo em atividades relacionadas ao desenvolvimento e manutenção de software.

Em abril de 1991, foi aprovado pelo comitê ANSI FORTRAN X3J3 (ISO 1539:1991) o padrão FORTRAN 90 [FORT92], que inclui o FORTRAN 77 mais inovações. Muitas destas inovações já existem em alguns compiladores na forma de extensões à linguagem FORTRAN 77, como o VAX-FORTRAN [VAXF84b] e o Microsoft FORTRAN [MSFO89b], por exemplo. Entretanto, Metcalf e Reid [METC91] observam que, ao contrário do padrão anterior (FORTRAN 77), que resultou quase que exclusivamente de um esforço para padronização de *práticas existentes,* este novo padrão (FORTRAN 90) é muito mais um *desenvolvimento* da linguagem, introduzindo características que são novas para o FORTRAN, baseadas na experiência com outras linguagens. Uma destas novas características é a capacidade de definir e manipular tipos de dados definidos pelo usuário.

Como se pode observar a linguagem FORTRAN ainda continuará sendo usada, por um bom tempo, nas áreas científica e de engenharia pelo menos. Portanto, considera-se válida a iniciativa de desenvolvimento de técnicas que propiciem um aumento da produtividade e da qualidade no desenvolvimento

de novos sistemas nesta linguagem. Assim, através da identificação e seleção de programas que apresentem potencial de reutilização, em sistemas já existentes, pretende-se alcançar esses objetivos.

1.3 Objetivo da Tese

Esta tese apresenta e discute as questões centrais relacionadas ao conceito de reutilização de software, apresentando uma ferramenta para identificação e seleção de componentes de código FORTRAN reutilizáveis, a partir de um acervo de programas já existentes em um determinado ambiente, para povoar uma biblioteca de componentes reutilizáveis. Os programas usados para avaliação da ferramenta fazem parte do Sistema Integrado de Computadores de Angra (SICA), desenvolvido pela COPPE/UFRJ para auxiliar a operação da usina atômica Angra I. A ferramenta seleciona automaticamente candidatos à reutilização, a partir de uma série de parâmetros estabelecidos. O modelo adotado é uma composição dos modelos propostos por Caldiera e Basili [CALD91] e Prieto-Diaz [PRIE91a], para identificação de componentes de código candidatos à reutilização, e do modelo proposto por Rocha [ROCH87], para avaliação da qualidade de software. A ferramenta utiliza teoria de conjuntos *fuzzy* [ZADE73], [ZADE84], [ZADE88], onde conceitos abstratos ou subjetivos podem ser representados através de variáveis linguísticas.

1.4 Organização da Tese

Esta tese está organizada em cinco capítulos. O primeiro capítulo contém a Introdução, onde é enfatizada a importância da reutilização de software na atividade de desenvolvimento de software para obtenção de maiores e melhores índices de produtividade e qualidade respectivamente. São, também, comentados os papéis da Engenharia de Software e da Qualidade de Software na obtenção de estratégias viáveis para a reutilização de software. Discutem-se os motivos do uso da linguagem FORTRAN pela comunidade científica mundial e, finalmente, apresenta a forma de organização deste trabalho.

O Capítulo 2 apresenta um estudo das tecnologias de reutilização de software, traçando um perfil atualizado destas tecnologias, procurando definir conceitos relacionados à reutilização de software, discutir alguns pontos cruciais, descrever experiências passadas e apresentar perspectivas futuras para o sucesso das tecnologias de reutilização de software.

O Capítulo 3 apresenta uma revisão da literatura sobre o atributo reutilizabilidade. São apresentadas as características de reutilizabilidade e os critérios para a identificação de componentes de código reutilizáveis.

No Capítulo 4 é proposto um conjunto de critérios para avaliação da reutilizabilidade de componentes, segundo o modelo de qualidade proposto por Rocha [ROCH87]. Parte-se da revisão da literatura e dos trabalhos desenvolvidos por Andrade [ANDR91] e Belchior [BELC92] nesta área. A seguir é feita uma exposição da teoria de conjuntos *fuzzy,* através de seus conceitos básicos, e são apresentados os conceitos de variável linguística e termo linguístico. Ao final, faz-se uma descrição da ferramenta ReFOR e de como esta pode ser utilizada.

Por fim, o Capítulo 5 relata algumas conclusões de caráter geral obtidas através da discussão dos resultados obtidos no experimento, sua contribuição e as perspectivas de continuidade deste trabalho.

2 REUTILIZAÇÃO DE SOFTWARE

2.1 Conceitos e Definições

A idéia de reutilização existe desde que os primeiros seres humanos começaram a resolver problemas. Quando resolvemos um problema, tentamos aplicar a solução a novos problemas similares. Se porventura constatamos que somente alguns elementos da solução se aplicam, nós os adaptamos para resolver o novo problema. Soluções comprovadas, usadas repetidamente para resolver o mesmo tipo de problema, são aceitas, generalizadas e padronizadas. Os modelos matemáticos são um bom exemplo e comprovam o sucesso da reutilização em várias áreas da engenharia.

Segundo o dicionário Aurélio da Língua Portuguesa, o significado técnico para reutilização é "procedimento em que material que já foi anteriormente processado, após o tratamento conveniente, se insere numa corrente de processo". Podemos, portanto, observar que o termo reutilização é bastante amplo e abrangente.

Tracz em [TRAC90] discute quais os aspectos mais relevantes para se criar uma estratégia de reutilização. Ele segmenta esta discussão em três elementos distintos: *produto*, ou o que se quer reutilizar, *processo,* ou como e quando reutilizar, e *pessoal*, ou quem faz a reutilização acontecer.

Na construção de programas, a reutilização de software é parte fundamental do processo, apesar do termo reutilização não ser freqüentemente explicitado. São exemplos disso a inclusão de macros em programas, que são trechos de código comuns a vários programas; as bibliotecas de rotinas em código objeto, que são extensões à linguagem para aplicações específicas (bibliotecas de rotinas matemáticas, gráficas, de interface, etc.).

Um exemplo de reutilização é a própria programação. Desde o início da programação, os programadores têm reutilizado código, rotinas, programas e algoritmos. Eles também sabem como adaptar e reverter sistemas. Mas tudo isto é feito de maneira informal [PRIE93].

No simples fato de um engenheiro de software usar sua experiência no projeto de novos sistemas, podemos encontrar uma forma poderosa de reutilização, ele está reutilizando o conhecimento adquirido em experiências anteriores [BARN91], [BASI92].

Uma definição para reutilização de software nos é fornecida por Bersoff [BERS91]. Segundo ele, a reutilização de software consiste no processo de incorporação, em um novo produto, de qualquer um dos seguintes elementos: código previamente testado, projetos previamente experimentados, especificações de requisitos previamente desenvolvidas, ou planos de testes e procedimentos previamente usados. O efeito da reutilização de software é uma redução considerável no tempo e custo de desenvolvimento, e um aumento da confiabilidade, uma vez que os componentes reutilizados já foram previamente usados e experimentados.

Apesar do conceito de reutilização já estar amplamente difundido, e o seu potencial reconhecido e aceito por aumentar a qualidade, confiabilidade e produtividade do software, o que falta para que a reutilização se torne uma realidade?

Para Prieto-Díaz não existe falta de reutilização, existe sim uma falta de formalismos e de uma sistemática ampla de reutilização [PRIE93].

Alguns autores [BIGG87], [FRAK91], [PRIE91a], [PRIE93], [TRAC88a], [TRAC88b], [TRAC90] afirmam que os problemas relacionados a reutilização não são meramente técnicos. Os problemas estão relacionados a fatores de ordem gerencial, econômica, cultural, psicológica e legal. Alguns destes problemas são tão importantes, e talvez mais difíceis de resolver, quanto os problemas técnicos. A visão atual de reutilização tende para uma integração de todos estes fatores num conceito de reutilização "institucionalizada" [PRIE93].

Os resultados de sucesso alcançados por algumas empresas japonesas e americanas, relatados em [PRIE91a] e [PRIE93], demostram que uma abordagem correta e realista destes fatores não-técnicos em conjunto com os fatores técnicos, permitem alcançar os resultados desejados para a reutilização.

Do ponto de vista psicológico, é vital a motivação dos profissionais, de todos os níveis, envolvidos no processo de reutilização. A implantação de uma nova metodologia de trabalho (a reutilização) implica numa mudança de hábitos, que geralmente não é bem recebida, daí a necessidade de elementos que motivem estes profissionais. Por exemplo, a participação de todos através de discussões e sugestões, por si só, já é um fator de motivação, pois o indivíduo está participando.

Do ponto de vista cultural, é imprescindível a criação de uma cultura voltada para a reutilização. Portanto, é necessário que todos os profissionais envolvidos recebam treinamento e material de trabalho adequados.

Quanto aos problemas de ordem econômica, são necessárias ferramentas que permitam ao gerente calcular o custo/benefício alcançado com a nova tecnologia, comparativamente ao processo convencional (sem reutilização). Estas medidas são importantes, pois o investimento inicial para a produção de software reutilizável é maior do que o investimento num processo convencional.

Do ponto de vista técnico, os problemas estão intimamente relacionados à tecnologia escolhida. Estes problemas têm praticamente concentrado todo o esforço atual de pesquisa na área de reutilização.

Quanto aos aspectos legais, são atualmente os pontos mais controvertidos e apresentam-se como barreiras à reutilização. Estes fatores estão estreitamente associados a fatores de ordem social, cultural e política e apresentam questões difíceis, tais como aspectos relacionados a direitos autorais e de propriedade, obrigações e responsabilidades [PRIE91a].

2.2 O Processo de Reutilização

O processo para reutilização de código existente é composto por quatro etapas: definição, recuperação, adaptação e incorporação [OSTE92]. A etapa de definição descreve o componente que precisamos construir nos termos de funcionalidade e relações dentro das especificações do sistema em desenvolvimento, onde o componente será usado. Portanto, antes de procurar temos que saber o que procurar. A etapa de recuperação consiste em tomar a definição para pesquisar o repositório, e recuperar uma lista de componentes com características similares (candidatos) e escolher um destes. A etapa seguinte (adaptação) tem por objetivo adaptar o novo componente. Na última etapa (incorporação), o componente novo está pronto para ser incorporado ao sistema em desenvolvimento .

O objetivo da reutilização de software, a longo prazo, depara-se com um conflito de projeto inerente: para ser útil, um sistema construído através de reutilização deve usar vários blocos de construção disponíveis, mas quando muitos blocos de construção estão à disposição torna-se um problema, de difícil solução, achar e escolher um apropriado. Um *sistema computacional de alta funcionalidade* suporta a reutilização através do fornecimento de muitos objetos que podem ser chamados ou adaptados. Os objetos de software podem ser funções, classes (no caso de orientação a objetos), programas inteiros ou fragmentos de código. Quanto maior e mais diverso o número de objetos, maiores serão as dificuldades (de recuperação) e as chances de que o objeto satisfaça completamente às necessidades de um determinado problema. Num *sistema computacional de baixa funcionalidade* os objetos são facilmente encontrados, mas as chances de que eles satisfaçam completamente as necessidades de um determinado problema são muito pequenas [FISC91].

A reutilização de software não pode ficar restrita à reutilização de código, apesar de ser a primeira idéia a surgir quando se fala de reutilização de software. A reutilização de software deve abranger todo o amplo espectro do processo de desenvolvimento de software. A característica chave de uma reutilização de software efetiva, na sua essência, implica na reutilização da habilidade de resolver problemas [BARN91]. Portanto, o conceito de

reutilização de software deve ser amplo e abrangente de maneira que admita a reutilização de "qualquer coisa" existente relacionada a software.

A reutilização de especificações e projetos é menos comum do que a reutilização de código, apesar de não ser menos importante. A reutilização de código é uma das técnicas de produtividade mais poderosas que existem, apesar de não ser a mais efetiva em termos de custos. Companhias como Toshiba e Hartford Insurance são capazes de desenvolver novas aplicações com até oitenta e cinco por cento de código proveniente de fontes reutilizáveis. Um estudo desenvolvido por Barry Silverman sobre código reutilizado na NASA verificou que para os gerentes o volume de código reutilizado nos projetos estudados totalizava menos de cinco por cento, enquanto que os programadores disseram que esse volume era de aproximadamente sessenta por cento [JONE91].

A identificação de características de reutilização implica na decomposição de sistemas de software no conjunto de seus conceitos elementares, a partir do qual podem ser extraídos os conceitos com potencial de reutilização identificados. Quando o número desses conceitos permitir a criação de conjuntos de conceitos específicos semelhantes, pode-se então abstrair um conceito genérico do elemento de software. Estes elementos são, então, classificados e catalogados em um repositório, estando assim disponíveis para reutilização.

A reutilização de código deve ser usada em um grande número de empreendimentos. Ela deve ser considerada na análise de produtividade em grandes empresas e empreendimentos com mais de duzentos programadores e analistas. Certos tipos de empresas são fortes candidatas à reutilização, alguns exemplos são empresas das áreas aerospacial, bancária, computadores, eletrônicas, comunicações, seguros, utilidade pública e as companhias telefônicas. Estas empresas desenvolvem um grande número de aplicações similares, e é exatamente nestes casos onde a reutilização é mais efetiva, pois tratam de domínios de aplicações específicas [JONE91].

2.3 Tecnologias de Reutilização

A questão da reutilização de software pode ser abordada de forma variada a partir de uma série de pontos de vista. Cohen, por exemplo, apresenta três modelos para o processo de reutilização de software [COHE91] :

Adaptação é o processo através do qual é feita a adaptação de produtos de software de um determinado domínio, bem definido e com mutabilidade lenta, permitindo a criação de novos sistemas a partir dos sistemas já existentes. Esta abordagem traz grandes benefícios desde que as novas modificações sejam incrementos aos produtos de software existentes.

Parametrização é o processo através do qual todas as novas implementações de sistemas partem de uma estrutura genérica para o domínio em questão. O retorno é significativo, desde que a estrutura seja estável.

Engenharia de Reutilização é o processo através do qual os recursos do domínio devem atender a um amplo espectro de tipos de aplicação do domínio. Se os recursos de domínio foram bem desenvolvidos este modelo apresenta grande flexibilidade e adapta-se muito bem a domínios com mutabilidade rápida.

Biggerstaff e Richter [BIGG87] fazem uma avaliação da reutilização de software sob o aspecto da tecnologia envolvida. Eles identificam e avaliam os processos através dos quais se faz a reutilização, segundo a natureza dos componentes envolvidos, e identificam dois grupos principais de tecnologias de reutilização: composição e geração.

Prieto-Diaz [PRIE93] apresenta uma taxonomia da reutilização cujas facetas, ou perspectivas, são mostradas na Tabela 2.0. Esta classificação por facetas conceitualmente independentes facilita a exposição do estado atual da pesquisa, da arte e da prática da reutilização de software.

Apesar desta taxonomia apresentar as perspectivas (facetas) de forma aparentemente independentes elas se misturam e entrelaçam. Ainda assim, esta estrutura de classificação mostra-se apropriada para uma ampla

apresentação, permitindo uma discussão setorizada dos problemas e abordagens da área de reutilização de software.

Substância	Escopo	Modo	Técnica	Intenção	Produto
Idéias, conceitos	Vertical	Planejado, sistemático	Composição	Caixa-preta, como está	Código fonte
Artefatos, componentes	Horizontal	Ad-hoc, oportunista	Geração	Caixa-branca, modificável	Projeto
Procedimentos, experiências					Especificações
					Objetos
					Texto
					Arquiteturas

Tabela 2.0 - Facetas da Reutilização de Software [PRIE93].

2.3.1 Natureza da Reutilização

A natureza da reutilização pode ser dividida em reutilização de idéias e conceitos, artefatos e componentes, procedimentos e experiências.

Reutilização de Idéias

Este tipo de reutilização envolve a reutilização de conceitos formais, tais como soluções gerais para uma determinada classe de problemas. Algoritmos genéricos são um bom exemplo deste tipo de reutilização. O atual estado de pesquisa, da arte e prática do desenvolvimento de algoritmos está relativamente maduro. Entretanto, da perspectiva de reutilização é ainda necessário o desenvolvimento e distribuição de catálogos padrão com informações específicas e detalhadas de como proceder para a reutilização de algoritmos [PRIE93].

Reutilização de Artefatos

Este é o ponto sobre o qual têm sido concentrados os esforços de pesquisa de reutilização de software nas últimas décadas. As fábricas de software [FERN92], [SWAN91], [BASI92] e os programas de reutilização em grandes corporações [TRAC90], [PRIE91a], [MATS87], [FRAK91] são exemplos concretos do esforço despendido para alcançar o sucesso na reutilização de software, através do aumento da produtividade, qualidade e confiabilidade. Atualmente as técnicas de orientação a objetos detêm as melhores perspectivas para reutilização de componentes - este tema é tratado no capítulo referente a programação orientada a objetos e a reutilização de software.

O esforço de pesquisa na área de componentes de software reutilizáveis está atualmente concentrado no desenvolvimento de uma tecnologia para avaliação da qualidade e confiabilidade de componentes reutilizáveis, no sentido de se obter um certificado de qualidade e uma homologação destes componentes [DAVI91], [HISL91], [PATE91], [PATE92], [YGLE92].

Fábricas de Software O conceito de fábrica de software apresenta o desejo de uma evolução do paradigma de produção de software artesanal para um estilo de produção mais industrializado, no qual investimentos substanciais podem ser feitos dentro de uma margem de risco aceitável.

O modelo clássico de uma fábrica, onde as pessoas agem como máquinas na execução de tarefas predeterminadas e repetitivas não se aplica a uma fábrica de software. No contexto do software, a analogia da fábrica pode ser aplicada somente com o propósito de estabelecer um estilo de produção industrial, não a sua implementação. A manufatura de software envolve pouco ou nada da produção tradicional: cada sistema é único. A maioria dos ambientes de desenvolvimento de software enfatiza o suporte para produção de código e documentação associada. Numa fábrica de software, o foco desloca-se para a coordenação das informações entre produtores e consumidores, de forma que a pessoa certa tenha sempre a informação certa no tempo certo [FERN92].

Segundo Cusamano, em [SWAN91], é importante observar que o termo *fábrica de software* vêm sendo usado desde a década de sessenta e, apesar do seu significado variar de país para país e mesmo entre organizações diferentes de um mesmo país, ele engloba a maioria das ferramentas CASE e técnicas de produção de software desenvolvidas.

Uma das principais características das fábricas de software é a importância que é dada às informações acumuladas de vários projetos. Estas informações são de vários tipos: elementos reutilizáveis (código, projeto, documentação, etc.), medidas de desempenho, processos de desenvolvimento e relatórios da efetividade da aplicação de técnicas específicas [FERN92].

Uma fábrica de software deve ter os meios necessários para descrever e analisar o conhecimento e seu contexto, meios eficientes de armazenamento, recuperação e compartilhamento deste conhecimento, para poder aplicá-lo a novas situações. Além disso, o conceito de fábrica de software deve englobar evolução, ou seja, a tecnologia atual deve ser capaz de coexistir com a tecnologia de amanhã [BASI92].

Um exemplo de fábrica de software é o projeto EUREKA SOFTWARE FACTORY (ESF) [FERN92] que desenvolveu uma arquitetura CASE centralizada em comunicação, que combina as vantagens de ambientes altamente interativos com flexibilidade evolucionária e capacidade de abrangência global do ambiente. Um ambiente ESF é composto por um conjunto de componentes, onde cada um destes componentes é composto de um grupo de ferramentas CASE totalmente integradas. Os ambientes ESF são construídos em torno de um *barramento de software*, que oculta a distribuição e heterogeneidade dos componentes conectados a ele. Os componentes são ilhas integradas que operam entre si através do barramento de software. O modelo do processo de produção de software é expresso em uma linguagem formal com semântica executável - na forma de um *programa de processo* - que roda ao mesmo tempo em que o ambiente é usado, permitindo assim a automação da logística de informação da fábrica. Por exemplo, quando um usuário termina uma tarefa, dentro de um determinado contexto de trabalho, todos os outros membros da equipe são notificados pelo programa de processo. O programa de processo instanciado pode também acumular

experiência, automaticamente, através do recolhimento, classificação e armazenamento de informações do contexto em questão.

Uma abordagem de organização orientada à reutilização é apresentada por Basili em [BASI92]. Ele propõe uma estrutura organizacional que separa atividades de projeto específicas das atividades de encapsulamento para reutilização, com modelos de processos distintos para cada uma destas atividades. A estrutura proposta define duas organizações separadas: uma *organização de projetos* e uma *fábrica de experiência*. A organização de projetos é orientada aos projetos e seu objetivo é entregar os sistemas encomendados pelos clientes. A fábrica de experiência tem por objetivos monitorar e analisar o desenvolvimento de projetos, desenvolver e empacotar experiência para reutilização na forma de conhecimento, processos, ferramentas e produtos, e fornecê-los à organização de projetos quando requisitados. Flexibilidade e aperfeiçoamento contínuo são características identificadoras deste ambiente, pois mudanças na configuração não têm impacto negativo sobre o desempenho, e a organização é capaz de aprender com suas experiências e evoluir para maiores níveis de qualidade, através do crescimento de sua capacidade e sua reutilização.

Certificação de componentes Um dos principais obstáculos para a produção de componentes de software reutilizáveis, com altos níveis de qualidade e confiabilidade, é a falta de uma tecnologia sistemática para análise destes componentes. Esta tecnologia sistemática é necessária para certificar a qualidade e adequação dos componentes de software reutilizáveis [PATE92].

Davis em [DAVI91] traduz a certificação de um componente de software como sendo a garantia de que estes componentes de software implementam seus requisitos e que a execução deles estará livre de erros no ambiente para os quais foram projetados.

Patel [PATE92] complementa que o objetivo do Certificado de Componente Reutilizável é garantir segurança ao usuário de componentes de software reutilizáveis. Ele apresenta, ainda, algumas definições preliminares de John Knight referentes a tópicos de certificação de componentes reutilizáveis:

***Propriedade**: um atributo característico que deve representar alguma afirmação verdadeira sobre um componente de software reutilizável.*

***Padrão de Certificação**: um conjunto de propriedades que um componente reutilizável pode possuir.*

***Componente Certificado**: um componente de software que demonstrou possuir as propriedades mencionadas num determinado padrão de certificação. Portanto, um componente de software pode ser certificado em relação a um ou mais padrões de certificação.*

***Tecnologia de Certificação**: o processo através do qual deve ser submetido um componente de software para obter uma certificação.*

A certificação está intimamente ligada ao vocabulário de termos de reutilização de software empregado. O vocabulário deve ter um número de termos amplo o suficiente que permita descrever o vasto campo de idéias que o usuário possa querer transmitir. Os termos deste vocabulário devem ter o mesmo significado para autores e leitores (e desenvolvedores e usuários). Portanto, estes termos devem ser definidos de forma clara e objetiva [PETE91].

Yglesias em [YGLE92] afirma que o conhecimento e concordância com padrões de certificação, para um determinado programa de reutilização em evolução, não são condições suficientes para o sucesso da obtenção de componentes reutilizáveis. Segundo a autora, a falta de uma terminologia padrão, principalmente uma terminologia para classificação dos componentes, e a falta de uma educação voltada para a cultura da reutilização são obstáculos ao sucesso da implantação do programa de reutilização de software.

O estabelecimento de padrões de certificação é um campo que requer maior esforço de pesquisa, já que diferentes domínios de aplicação, requerem padrões de certificação diferentes, e a terminologia da reutilização de software requer uma padronização, uma linguagem comum.

Prieto-Diaz [PRIE93] observa que outro ponto importante de pesquisa atualmente é a adaptabilidade de componentes e cita como exemplo de reutilização de software, dentro de um domínio de aplicação específico, o projeto CARDS (Central Archieve for Reusable Defense Software), descrito em [WALL92]. O objetivo deste projeto é avaliar a transição da tecnologia de reutilização de software de domínio específico centralizado em uma biblioteca, através do uso de técnicas de representação do conhecimento para a modelagem da biblioteca, no sentido de permitir descrever e gerenciar o grande número de relacionamentos que existem entre os componentes da biblioteca. O modelo da biblioteca é expresso através de um sistema híbrido composto por uma rede de restrições e regras de inferência.

Reutilização de Procedimentos

A reutilização de procedimentos, segundo Prieto-Diaz [PRIE93], é uma das áreas de pesquisa mais intensas atualmente. A pesquisa está voltada para a possibilidade de se obterem conjuntos de processos reutilizáveis que possam ser interligados para que se obtenham instâncias de processos novos e mais complexos.

A reutilização de processos é um dos principais objetivos do projeto STARS [DAVI91], [DAVI92]. A estrutura conceitual para reutilização de processos do STARS descreve um conjunto de processos reutilizáveis que podem ser combinados para a implementação de programas de reutilização particulares.

2.3.2 Domínio de Aplicação

Podemos definir domínio de aplicação como uma área de aplicação, ou esfera de atividades, para a qual são desenvolvidos sistemas de software [PRIE90]. São domínios de aplicação, por exemplo, operações financeiras, transações bancárias, controle numérico, monitoração e controle, etc. Os domínios podem ser amplos como o de operações financeiras ou restritos como o das operações matemáticas.

O conceito de domínio de aplicação é de suma importância para a reutilização de software. O domínio de aplicação surge a partir da análise de domínio. A análise de domínio foi introduzida por Neighbors [NEIG89] como sendo a atividade responsável pela identificação dos objetos e operações de uma classe de sistemas similares em um domínio particular de problemas. Para Neighbors, a chave para reutilização de software é obtida na análise de domínio, na qual enfatiza a reutilizabilidade de análise e projeto, não do código.

Prieto-Diaz também considera que o caminho para o sucesso da reutilização de software está baseado numa análise de domínio bem feita. Segundo ele, análise de domínio é o processo através do qual a informação usada no desenvolvimento de software é identificada, extraída e organizada para ser reutilizada na criação de novos sistemas [PRIE87b].

Para Tracz [TRAC92] a análise de domínio é uma área de pesquisa relativamente nova e ainda não existe um consenso para sua definição, nem uma descrição do processo para executá-la.

Para se desenvolver um programa aplicativo são necessários o conhecimento de programação e o conhecimento do domínio da aplicação. O conhecimento de programação é relativamente bem conhecido, pois é formal, pode ser modelado de várias maneiras e genérico o suficiente para ser aplicado numa ampla variedade de domínios. O conhecimento do domínio, apesar de existir na mente dos especialistas do domínio, não é tão bem conhecido. Normalmente ele é informal, implícito, específico e modelado de forma incompleta e indireta, através de linguagens específicas para determinados problemas [ISCO91].

Na análise de domínio são identificados objetivos e operações comuns a todos os sistemas dentro do mesmo domínio, e um modelo é definido para descrever seus relacionamentos. Os componentes resultantes desta análise são mais apropriados à reutilização porque eles capturam a funcionalidade essencial requerida no domínio [FARI91]. Apesar desta abordagem aparentemente desconsiderar a reutilização de componentes de utilidade genérica, isto não significa que eles não devam ser reutilizados. Geralmente, o custo destes componentes é maior do que o benefício proporcionado. Para Neighbors estes

componentes pertencem a domínios de modelagem [NEIG89]. Eles encapsulam o conhecimento de engenharia necessário para o desenvolvimento de sistemas. Do ponto de vista hierárquico, os domínios de modelagem estão num nível abaixo dos domínios de aplicação. O conhecimento obtido do domínio de aplicação agrega objetos dos domínios de modelagem [FARI91].

Reutilização Vertical

Reutilização vertical é a reutilização dentro do mesmo domínio ou área de aplicação. Seu objetivo é derivar módulos genéricos para famílias de sistemas que podem ser usados como modelos para a construção de novos sistemas. Quanto mais restrito o domínio, maior o benefício (de retorno). O foco de pesquisa na reutilização vertical tem sido concentrado sobre a análise de domínio. Vários métodos de análise de domínio vêm sendo desenvolvidos nos últimos anos. Um exemplo é o Feature Oriented Domain Analisys [COHE91], [BAIL92]. Vários métodos de orientação a objetos também vêm sendo estendidos, com o objetivo de abranger a análise e modelagem de domínios. Apesar da reutilização vertical estar sendo praticada informalmente na maioria das organizações de software, a tendência definitiva é alcançar uma prática sistemática. As fábricas de software, na sua maioria, estão interessadas na reutilização vertical. Organizações com projetos de vida longa, tais como a NASA e o Departamento de Defesa dos Estados Unidos da América, estão concentrando seus esforços na reutilização vertical. Pacotes de software especializados como o Motif são um exemplo de sucesso da reutilização vertical [PRIE93].

Reutilização Horizontal

Reutilização horizontal é a reutilização de partes genéricas em diferentes tipos de aplicações. Bibliotecas de rotinas científicas e as ferramentas do sistema operacional UNIX são um bom exemplo de reutilização horizontal. Apesar da pesquisa em reutilização horizontal estar concentrada no empacotamento e apresentação de partes componentes, o foco atual de pesquisa está concentrado no desenvolvimento de bibliotecas de amplo acesso e em

bibliotecas em rede. Projetos como o Asset Source for Software Engineering Technology e o Center for Software Reuse Operation, são exemplos de iniciativas para o desenvolvimento de tais repositórios. Estes projetos estão alcançando progresso significativo no estabelecimento de padrões de catalogação e interoperabilidade e repositórios em rede.

Na medida em que estas iniciativas forem se desenvolvendo, uma base industrial de partes de software de reutilização vertical e horizontal vai emergir [PRIE93].

Engenharia de Domínio

A análise de domínio é sem dúvida a chave para uma reutilização sistemática, formal e efetiva, e é através dela que o conhecimento é transformado em artefatos genéricos: especificações, projetos, arquiteturas, etc. Estruturas genéricas formarão a base para a criação de componentes de fácil reutilização [PRIE93]. Uma definição para engenharia de domínio é apresentada em [PETE91] como "a construção de componentes, métodos e ferramentas e respectiva documentação para solucionar os problemas de desenvolvimento de sistemas através da aplicação do conhecimento contido no modelo de domínio e nas arquiteturas de software". Segundo Arango [ARAN89], o propósito da engenharia de domínio é desenvolver métodos e representações que auxiliem na: 1) construção e análise de modelos de domínios de problemas para melhor suportar a especificação e a construção de software; e 2) a tranformação desses modelos com o propósito de melhorar o desempenho dos sistemas de reutilização de software.

2.3.3 Modos de Reutilização

Reutilização Planejada

A reutilização planejada é a prática sistemática e formal da reutilização, onde são definidas estratégias e procedimentos para reutilização de software. São

também necessários, vários tipos de métricas que permitam fazer uma avaliação global do desempenho da reutilização. Este tipo de reutilização é normalmente encontrado nas fábricas de software [PRIE93].

Para se implementar um programa de reutilização planejada também são necessárias mudanças significativas na prática corrente de desenvolvimento de software, disciplina e comprometimento de todo o pessoal envolvido, incentivos à prática da reutilização, métodos gerenciais de avaliação e controle e modelos econômicos para a estimativa de custos.

Prieto-Diaz, em [PRIE91a], enumera uma série de fatores que afetam a reutilização e apresenta um modelo incremental para implementação de um programa de reutilização, onde esses fatores são tratados a medida que vão surgindo durante o processo de reutilização. Esta abordagem permite que a organização inicie um programa de reutilização e aprenda com ele.

Um dos problemas da reutilização planejada é a falta de modelos econômicos. Os gerentes precisam ter meios de saber quais são os benefícios, retorno do investimento, custo inicial e critérios de desempenho, antes de se comprometerem com um programa de reutilização em grande escala [PRIE93].

Para Barnes [BARN91], a reutilização de software, no referente a investimentos, tem as mesmas características de custo e risco de qualquer aplicação financeira. Modelos econômicos para ciclos de vida de software reutilizável foram propostos por [BALD90] e [BARN91], mas precisam ser validados.

Reutilização Oportunista

A reutilização oportunista é a prática informal na qual os componentes são escolhidos em bibliotecas de uso genérico. A reutilização *ad-hoc* é o estado da prática: A reutilização é feita a nível individual e não a nível de projeto; não existem procedimentos para reutilização; e as bibliotecas usadas são compostas por componentes que não foram projetados para serem reutilizáveis [PRIE93].

A pesquisa na área da reutilização oportunista, ou ad-hoc, tem se concentrado no desenvolvimento de melhores bibliotecas de componentes reutilizáveis, com interfaces amigáveis e mecanismos de recuperação mais poderosos. São exemplos os trabalhos desenvolvidos por Prieto-Diaz [PRIE87a] e [PRIE91b], por Maarek [MAAR91], por Fischer [FISC91], por Isoda [ISOD92], por Ostertag [OSTE92] e por Wallnau [WALL92].

O principal problema encontrado aqui diz respeito ao esforço necessário para se povoar estas bibliotecas, pois catalogar e classificar componentes reutilizáveis continua sendo uma atividade manual cara, pois são necessários especialistas dos domínios envolvidos para executar esta tarefa [BIGG89a], [CALD91].

Caldiera [CALD91] apresenta um esquema semi-automático para seleção e classificação de componentes candidatos à reutilização. A primeira fase é automatizada e é responsável pela extração de componentes candidatos à reutilização, com base em métricas de software, a partir do acervo de programas de uma dada organização. Na segunda fase, o processo é manual, um especialista do domínio complementa as informações necessárias para catalogação e classificação dos componentes.

2.3.4 Técnicas de Reutilização

Reutilização por Composição

Este grupo de tecnologias caracteriza-se pela composição de aplicações a partir de componentes atômicos, bem definidos, idealmente, não modificados ao serem reutilizados [BIGG87]. A reutilização por composição, segundo Prieto-Diaz [PRIE93], consiste numa construção de blocos (componentes existentes) para a obtenção de novos sistemas. Ela está baseada em conjuntos de componentes bem estabelecidos, sistemas de bibliotecas eficientes e interfaces padronizadas.

Apesar da reutilização por composição advogar a reutilização de qualquer tipo de componente de software, na prática ela está, quase que exclusivamente, concentrada na reutilização de código [PRIE93].

O ambiente Eiffel [MEYE88] reflete o estado da arte da reutilização por composição na orientação a objetos. Em ambientes para prototipagem, o ambiente REBOOT [FAGE92] demonstra as tendências no sentido de formalizar a reutilização por composição, pois utiliza métodos associados para suportar a reutilização através da tecnologia de orientação a objetos.

Um ambiente de reutilização baseado no uso de bibliotecas de componentes deve prover um esquema adequado para a classificação de componentes, assim como ferramentas de suporte para a recuperação, compreensão, modificação e, principalmente, composição destes componentes [WERN92a].

O esquema de classificação é a chave para uma recuperação rápida e eficaz de componentes em uma biblioteca. A solução proposta por Prieto-Diaz [PRIE87a], [PRIE89] é um esquema de classificação facetado. Neste esquema as relações genéricas entre os componentes são descritas e classificadas a partir da síntese de suas classes elementares. Medidas de distância conceitual entre os termos do vocabulário usado permitem estabelecer o grau de relacionamento entre componentes. Outras abordagens de esquemas de classificação similares para a recuperação de componentes relacionados foram desenvolvidas por Maarek em [MAAR91] e por Ostertag e outros, em [OSTE92].

No caso da necessidade de adaptação do componente a novos requisitos, temos um aumento do potencial de reutilização na medida em que aumenta a generalidade do mecanismo. Para adaptação de um componente, é necessário um esforço mental para compreensão do modelo computacional do componente. Portanto, informações adicionais necessárias (texto, fluxos, desenhos, etc.) para o perfeito entendimento do componente devem estar disponíveis ao usuário, na biblioteca.

A adaptação, pura e simples, do componente seria o procedimento ideal caso não existisse a frequente necessidade de modificá-lo. A principal dificuldade nesta etapa do processo consiste na avaliação do quanto o componente

atende às necessidades do usuário e na sua capacidade de adaptá-lo aos requisitos. Este processo de adaptação depende exclusivamente do usuário e não existe, na prática, nenhum tipo de auxílio substancial.

Reutilização por Geração

Este grupo de tecnologias é de difícil caracterização uma vez que os componentes reutilizados não são facilmente identificáveis como entidades concretas. Na maioria das vezes, os componentes reutilizados são padrões embutidos no processo de geração [BIGG89a].

As tecnologias de reutilização por geração caracterizam-se, principalmente, pela sua natureza declarativa, partindo-se de uma descrição do problema até se alcançar sua implementação, sendo este processo realizado de forma automática ou semi-automática [WERN92a].

Exemplos de tecnologias de reutilização por geração são os geradores de aplicações [CLEA88], linguagens de quarta geração [PRES92] e sistemas baseados em transformações [BIGG89a].

Geradores de Aplicação Transformam especificações em programas aplicativos. Esta categoria de sistemas apresenta um grande potencial de reutilização de software. Para domínios específicos, o conhecimento (sobre este domínio) está contido no projeto do gerador. A cada nova aplicação gerada os padrões contidos no projeto do gerador são reutilizados na criação de um novo aplicativo. Este tipo de sistema adapta-se muito bem à domínios de aplicação com grande potencial de parametrização. Isto é verdade para aplicações de pequeno e médio porte, pois para grandes aplicativos provou ser ineficiente [HORO89].

A especificação usada pelo gerador de aplicação descreve o problema ou a tarefa a executar através do programa. Estas especificações podem ter a forma de um diálogo interativo, onde o usuário navega através de uma série de menus selecionando as opções desejadas. Pode também ter um formato gráfico, onde o usuário edita um ou mais diagramas. Pode ainda ser escrita

numa linguagem, algumas vezes chamada de linguagem de quarta geração ou linguagem orientada à aplicação. Independente da sua forma, a especificação é usada pelo gerador para criar automaticamente um ou mais produtos, normalmente um segmento de código, uma rotina ou um sistema de software.

Para se construir um gerador de aplicação, é necessário que o domínio de aplicação esteja bem definido e todos os projetistas envolvidos o conheçam profundamente. Conceitualmente, podemos considerar um gerador de aplicação como um resultado final de uma análise de domínio [HORO89].

O sistema DRACO [NEIG89] é um exemplo de gerador de aplicações. Este sistema requer o desenvolvimento de uma linguagem de domínio específica na qual o usuário pode especificar o seu problema. O sistema gera programas a partir de especificações de domínios específicos, e utiliza um conjunto de transformações definidas pelo usuário para concluir a geração.

Segundo Biggerstaff, quanto mais restrito o domínio de aplicação de um gerador maior e mais rápido é o benefício, ainda que isto implique numa reutilização limitada [BIGG89a].

2.3.5 Intenção da Reutilização

Caixa Preta

A reutilização tipo caixa-preta consiste na reutilização de componentes sem nenhuma modificação. Tipicamente, estes componentes reutilizáveis são empacotados e suas interações definidas através de interfaces padronizadas.

Apesar desta abordagem garantir altos padrões de qualidade e confiabilidade, o custo para se produzir componentes reutilizáveis caixa-preta é maior do que para componentes reutilizáveis modificáveis (*caixa-branca*) [PRIE93].

Para produção de componentes reutilizáveis caixa-preta, os conceitos de ocultação de informação e herança são conceitos chaves para a obtenção de componentes modulares e independentes.

Um ponto importante que deve ser ressaltado aqui é o problema de verificação e certificação, ou seja, como demonstrar que um componente vai executar sem erros sob todas as possíveis condições de uso. Segundo Prieto-Diaz [PRIE93], este problema gerou um volume significativo de esforço de pesquisa. Conforme estas pesquisas, uma possível saída para este problema seria o emprego de especificações formais especiais que poderiam ser usadas para comprovar a exatidão dos programas e, consequentemente, reduzir a necessidade de testes exaustivos.

Um exemplo de biblioteca de componentes reutilizáveis do tipo caixa-preta é relatado por Lenz [LENZ87], onde os usuários não podem modificar ou sequer visualizar internamente a concepção dos componentes.

Caixa Branca

A reutilização tipo caixa-branca consiste na reutilização de componentes através da modificação e adaptação destes componentes.

Segundo Prieto-Diaz [PRIE93] esta é sem dúvida a abordagem mais comum, principalmente quando a reutilização é do tipo oportunista (ad-hoc). A maioria dos programas de reutilização, inclusive as fábricas de software, utilizam esta abordagem. A medida que estes programas evoluem, a reutilização através da adaptação torna-se mais formalizada.

A tendência da reutilização de componentes do tipo caixa-branca está direcionada no sentido da parametrização, adaptabilidade embutida e na expansão da reutilização para outros produtos, como projetos e especificações. A análise de domínio tem um papel muito importante na identificação de como as famílias de sistemas variam [PRIE93].

Generalidade de Componentes A dimensão de generalidade de componentes introduz o conceito de poder relativo de uma tecnologia de reutilização, expresso em termos de quantas variações de uma parte podem ser prontamente obtidas através da aplicação desta tecnologia. A generalidade mede o quanto uma tecnologia particular cobre as necessidades de uma área de aplicação. Devido ao fato de poucas partes poderem ser reutilizadas sem alguma modificação, uma tecnologia que forneça grandes conjuntos de componentes "fáceis-de-reutilizar", com variantes pré-definidas, aumentam em muito o número total de oportunidades de reutilização. Mas esta construção de generalidade em partes reutilizáveis tende a ser cara e trabalhosa [BARN91].

Métodos generativos são um exemplo extremo da generalidade de componentes. Estes métodos permitem a síntese de partes reutilizáveis a partir de linguagens de aplicação específicas, quase da mesma forma que um compilador permite a síntese de módulos de código objeto a partir de linguagens de alto nível. Apesar dos métodos generativos fornecerem altos níveis de generalidade e facilidade de uso, eles requerem uma extensa análise e preparação. O alto custo de investimento, torna estes métodos apropriados somente quando se pode antecipar muitas instâncias de reutilização [BARN91].

2.3.6 Produtos Reutilizáveis

Na classificação apresentada na Tabela 2.0, a faceta por produtos é uma lista representativa, ainda que incompleta, dos tipos de produto de software que podem ser reutilizados.

Código Fonte

A reutilização de código fonte representa o estado da prática da reutilização de software. A maioria das ferramentas [BURT87], [WERN92b], ambientes [CALD91] e métodos [PRIE91a] tem por principal objetivo a reutilização de código.

Um dos obstáculos para a reutilização de código são as linguagens de programação. O esforço necessário para se converter um componente para uma outra linguagem de programação pode não compensar, pois esta nova linguagem pode ter características que compensem, ou necessitem, a criação de um "novo componente", em vez de uma conversão pura e simples.

Tamanho, estrutura e linguagem são fatores que limitam os benefícios alcançados pela reutilização de código. A experiência demonstra que menos da metade de um sistema qualquer pode ser construído a partir de componentes dessa natureza [BIGG87].

Projetos

A reutilização de idéias no processo de desenvolvimento de software corresponde ao reaproveitamento de informações de projeto. Os benefícios aqui são maiores porque o nível de abstração é maior e independente dos detalhes de implementação, consequentemente aumentando o potencial de reutilização.

As informações de projeto não devem conter muitos detalhes, a ponto de diminuir a reutilizabilidade do componente, e nem ser tão pouco detalhada que inviabilize a reutilização pelo maior esforço de adaptação requerido [FARI91]. O modelo de representação deve permitir graus controlados de abstração, ou seja, deve ser capaz de representar estruturas conceituais precisas mas livres de detalhes. A representação da essência do projeto é que permite aplicar conceitos a partir de um domínio para estruturas em um contexto inteiramente diferente.

O sistema DRACO [NEIG89] apresenta uma abordagem ambiciosa cujos objetivos são mais amplos, pois visa a reutilização a nível de definição, projeto e implementação.

Objetos

Os objetos vem ganhando terreno como os produtos de reutilização mais reutilizáveis. Existem vários métodos, ferramentas e ambientes para a criação de objetos. A tendência da orientação a objetos é no sentido de se integrar ferramentas e métodos para cobrir todas as fases do desenvolvimento. Métodos como Shlaer-Mellor [SHLA88], [LANG93] e Booch [BOOC91] cobrem todas as fases, da análise de domínio à programação, fornecendo as devidas ferramentas de suporte. A orientação a objetos parece ser a técnica do futuro para a reutilização de software [PRIE93].

As técnicas de orientação a objetos podem ser usadas para suportar, pelo menos parcialmente, a reutilização planejada, vertical e por composição.

Texto

Uma vez que todos os produtos, exceto código, são documentos para uso humano, a reutilização de textos está se tornando uma área de importância crescente. Apesar disso, está ainda em estágio de pesquisa, mas a abundância de conhecimento e técnicas de recuperação de informação prometem rápidos progressos [PRIE93].

As tecnologias de hipertexto também são fundamentais para a reutilização de textos.

Arquiteturas

Esta é a maior unidade (componente) de reutilização. Domínios de aplicação são analisados com o objetivo de se encontrar projetos genéricos, que são então usados como modelos básicos para a integração de componentes reutilizáveis ou para o desenvolvimento de geradores de código especializados. Esta área de pesquisa complementa e suporta ambas tecnologias de reutilização, por composição e por geração [PRIE93].

2.4 A Programação Orientada a Objetos e a Reutilização de Software

A programação orientada a objetos é um método de modelagem no qual os programas são organizados na forma de coleções cooperativas de objetos, onde cada um deles representa uma instância de alguma classe, e cujos objetos são todos membros de uma hierarquia de classes unidas através de relacionamentos de herança [BOOC91].

Segundo Booch [BOOC91], para uma linguagem ser realmente orientada a objetos ela deve:

- Usar *objetos* na sua lógica fundamental de blocos de construção;
- Cada objeto é uma *instância* de alguma *classe*;
- A relação entre as classes é estabelecida através de relacionamentos de *herança*.

Para Tracz [TRAC88a], o sucesso da reutilização tem origem na formalização de processos e produtos, criados por um ambiente orientado a objetos. Além disso, a reutilização de software não planejada é cara, o software deve ser projetado para ser reutilizado, e o paradigma da orientação a objetos esgota estas características.

Ainda segundo Tracz [TRAC88a], a adaptabilidade e o potencial de reutilização de um componente de software dependem do valor da análise de domínio executada e do grau de parametrização do componente para refletir esta análise de domínio. Além disso, o tipo das facilidades de parametrização disponíveis nas linguagens podem, as vezes, não suportar o grau ou forma de adaptabilidade desejados. Portanto, apesar de certas características da linguagem facilitarem o desenvolvimento de software reutilizável, a linguagem, por si só, não é suficiente para resolver o problema da reutilização.

As linguagens orientadas a objetos, entre todas as linguagens de programação são as que apresentam os maiores benefícios para reutilização de código [TARU88]. As razões para esta afirmação são:

- Os objetos estão mais de acordo com os conceitos que nós capturamos no reconhecimento do mundo real;

- Podemos reutilizar os tipos de dados e procedimentos através da reutilização de classes;

- Através da herança, as classes de propósito geral e as classes especializadas podem ser armazenadas em bibliotecas de classes. Os usuários podem reutilizar quantas classes especializadas forem necessárias;

- Classes podem ser facilmente modificadas através da criação de subclasses (especialização);

- Características de duas ou mais classes podem ser herdadas através de herança múltipla;

- Classes são modulares.

O desenvolvimento de software orientado a objetos (análise, projeto e programação) está baseado no princípio de que os problemas não nascem à partir de um mundo vazio. Portanto, o planejamento para reutilização é precisamente o problema de se ser capaz de tirar vantagem de um conjunto de experiências e recursos existentes e disponíveis de forma confiável e eficiente.

Uma grande vantagem do desenvolvimento de software orientado a objetos é a semelhança estrutural entre os artefatos nas diferentes fases do desenvolvimento. No contexto da reutilização, a vantagem reside no fato das modificações, geralmente, permanecerem locais ao longo das diferentes representações [KARL92].

Estudos do paradigma da orientação a objetos são importantes porque, segundo a estrutura para reutilização de software proposta por Biggerstaff [BIGG87], o paradigma da orientação a objetos tem uma relação balanceada entre potencial e generalidade. As soluções procedurais são apresentadas com uma boa relação de balanceamento, mas são consideradas menos efetivas do que as soluções orientadas a objetos.

As características fundamentais do paradigma da orientação a objetos parecem complementar as necessidades do desenvolvedor quanto à reutilização. A capacidade de encapsulamento permite criar objetos auto-contidos que podem ser facilmente incorporados em novos projetos.

Lea e Chapeaux [LEA92] não concordam com a afirmação de Tracz [TRAC88a], de que o suporte de ferramentas elaboradas e características da linguagem não contribuem para o sucesso da reutilização. Lea e Chapeaux reportam que esta afirmação não tem base experimental, no que diz respeito aos métodos de engenharia de software orientados a objetos, já que é grande a carência de resultados efetivos de experimentos controlados que evidenciem o aumento da produtividade da orientação a objetos no contexto da reutilização de software.

Tewari [TEWA92] realça a necessidade de se verificar e comprovar empiricamente as afirmativas de aumento da produtividade na reutilização de software através da orientação a objetos, em diferentes abordagens. São necessárias técnicas e abordagens para se estudar, na prática, o relacionamento entre reutilização de software e as técnicas de orientação a objetos. Cita, como exemplo, as premissas pesquisadas e os resultados empíricos obtidos por Lewis no experimento comentado a seguir.

Num trabalho pioneiro, Lewis [LEWI91] desenvolveu um experimento controlado, considerando uma linguagem procedural (Pascal) e outra orientada a objetos (C++). O experimento comparativo é dividido em duas partes, uma considerando a reutilização de código e outra não. O experimento concluiu que:

- o paradigma da orientação a objetos melhora consideravelmente a produtividade, embora uma parte significativa desta melhora seja devida a reutilização;

- a reutilização, independente do paradigma da linguagem, melhora a produtividade;

- a diferença entre o paradigma das linguagens é mais importante quando os programadores reutilizam código do que quando não reutilizam;

- o paradigma da orientação a objetos tem uma afinidade particular com a reutilização.

As principais características das linguagens orientadas a objetos que favorecem a reutilização de software são encapsulamento e herança [MEYE87].

Encapsulamento é o nome dado ao processo de ocultar as características internas de um objeto para suportar ou impor abstração [MSVC93a]. Isto implica numa distinção precisa entre a "interface" de um objeto e sua "implementação", já que a interface descreve o que ele pode fazer, enquanto que sua implementação descreve como ele faz. Devemos observar que encapsulamento e abstração são conceitos complementares.

A abstração está direcionada sobre a visão externa de um objeto e o encapsulamento previne que os usuários (clientes) tenham uma visão do seu interior, onde o comportamento da abstração está implementado [BOOC91]. A Figura 2.0 ilustra este conceito.

Herança é um relacionamento entre classes, dentro da qual uma classe compartilha a estrutura e/ou comportamento definido em uma (herança simples) ou mais (herança múltipla) classes. A herança define uma hierarquia do tipo "é um" entre as classes, na qual uma subclasse herda de uma ou mais superclasses; normalmente uma subclasse aumenta ou redefine a estrutura e o comportamento existentes de suas(s) superclasse(s) [BOOC91]. Ainda, segundo [COX87], herança é uma ferramenta para organizar, construir e usar classes reutilizáveis.

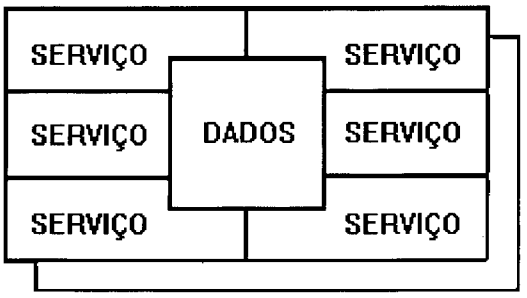


Figura 2.0 - Só podemos ter acesso aos dados de um objeto "instanciado" através dos serviços (métodos ou funções) disponíveis na classe do objeto.

Vantagens

O encapsulamento apresenta uma série de benefícios. Um deles é a abstração, já descrito, segundo a qual pode-se usar um módulo sem se saber como ele trabalha. Outro benefício é a localidade, quando alterações em uma parte do programa não implicam em alterações em outra(s) parte(s) do programa. Apesar do encapsulamento ser um dos elementos fundamentais do modelo de objetos, ele não é exclusivo a este modelo, é também um dos princípios básicos da programação estruturada. O benefício de localidade descrito, pode ser também descrito como um alto grau de coesão e baixo acoplamento (em se falando de módulos).

No que se refere a modularização, dois pontos técnicos podem afetar as decisões. Primeiro, uma vez que os módulos constituem as unidades elementares e indivisíveis de software que podem ser reutilizadas através das aplicações, o desenvolvedor pode optar por empacotar classes e objetos em módulos, de forma que tornem a sua reutilização mais conveniente. Segundo, muitos compiladores geram o código objeto em segmentos, um para cada módulo. Portanto, podem existir limites práticos para o tamanho dos módulos [BOOC91].

Os benefícios resultantes da herança podem ser resumidos em herança de código e herança de interface. Para se herdar o código de uma determinada classe, é necessário criar-se uma subclasse (especialização) e todo o código existente na classe estará "disponível" para a subclasse. No caso da herança de interface, o processo é praticamente o mesmo só que neste caso herdaremos só os nomes dos métodos e criaremos código novo para a subclasse. Teremos, portanto, a mesma interface da superclasse, que vai executar tarefas diferentes com funções (métodos) de mesmo nome.

A disponibilidade de herança em uma linguagem orientada a objetos diminui o tempo de desenvolvimento, aumenta os índices de confiabilidade e qualidade. A herança encoraja a maneira natural de organizar o conhecimento numa forma hierárquica. A organização hierárquica permite que a mente humana capture melhor e mais rapidamente o conhecimento [PINS90].

Outra vantagem apresentada pelas linguagens orientadas a objetos é o princípio de *dynamic binding* (ou *late binding*) [RINE91]. Segundo este princípio, os tipos de todas as variáveis e expressões só são conhecidos em tempo de execução. Este princípio, interativamente com o princípio de herança, permite o surgimento do *polimorfismo*, que é talvez a característica mais poderosa das linguagens orientadas a objetos [BOOC91]. O polimorfismo numa linguagem orientada a objetos traduz a capacidade de se enviar a mesma mensagem para instâncias de objetos de classes diferentes.

Desvantagens

Uma desvantagem potencial de algumas linguagens orientadas a objetos é a sobrecarga no processamento das mensagens. O processo de dynamic biding no processamento das mensagens, geralmente, consome mais tempo do que uma chamada (*call*) direta de função em outras linguagens [RINE91].

Outra possível desvantagem é a "distância semântica" entre as linguagens orientadas a objetos de alto nível e as máquinas nas quais elas rodam. Podem surgir problemas de performance no código desenvolvido, já que em geral as máquinas e os sistemas operacionais não têm uma arquitetura orientada a objetos [RINE91].

Uma desvantagem das linguagens orientadas a objetos é o tempo necessário para um programador (de linguagem convencional) aprender, dominar e lidar com a nova linguagem. Este tempo é grande se comparado a uma nova linguagem convencional. O aprendizado está baseado, principalmente, na inspeção de código e pesquisas na biblioteca de classes (nem sempre bem organizada), já que para se poder reutilizar o código existente é necessário saber que ele existe. Apesar disso, segundo Tesler [TESL86], os benefícios oferecidos pelas linguagens orientadas a objetos compensam em muito o tempo e esforço de aprendizado dispendidos.

O tempo necessário para o aprendizado de uma linguagem orientada a objetos depende muito das ferramentas disponíveis e da estrutura de implementação da biblioteca de classes. A estrutura destas bibliotecas deve ser composta de maneira que se possa estabelecer o que é básico para programação e o que não é. A longo prazo, a padronização de partes da biblioteca, como o sistema de janelas por exemplo, vai facilitar para os programadores o aprendizado de novos sistemas. As grandes bibliotecas são de valor inestimável, porque fornecem a funcionalidade que de outra forma seria implementada a partir do zero pelos programadores, a cada novo projeto [O'SHE86].

Outro problema de aprendizado refere-se ao domínio da aplicação, uma vez que o desenvolvimento orientado a objetos envolve tanto o aprendizado do domínio genérico quanto o do domínio específico, no que diz respeito ao domínio do problema proposto e ao espaço da solução, que deve ter por base bibliotecas de classes e bases de objetos. Portanto, a estratégia de ensino é influenciada pelo domínio particular da aplicação, no sentido de apontar os possíveis elementos do espaço da solução ao programador [RINE91].

As ferramentas disponíveis nas linguagens orientadas a objetos citadas anteriormente são, em geral, muito pobres. Resumem-se a *browsers* de classes e métodos com algum tipo de informação de referência cruzada, que é o caso do browser de classes do Smalltalk [GOLD83], do browser de classes e funções da linguagem Visual C++ 1.0 [MSVC93a], [MSVC93b] e do browser de classes e métodos da linguagem Actor [ACTO91]. Para bibliotecas de classes pequenas (básicas) funcionam bem, apesar de requerer um certo esforço de pesquisa. Mas no caso de bibliotecas médias e grandes (que são a maioria, já que elas crescem continuamente com a adição de novas contribuições) são de

pouco auxílio para a recuperação de informação sobre classes desconhecidas pelo programador.

Os trabalhos de [TARU88] e [LIAO93] apresentam propostas de ferramentas mais poderosas, permitindo pesquisas nas bibliotecas através de aspectos do contexto dentro de um determinado domínio. A pesquisa é feita através de consultas (*queries)* com termos chave, que em resposta retornam listas de classes ou métodos que contêm os referidos termos chaves pesquisados. A classificação das classes da biblioteca é feita a partir de um dicionário de termos chaves, definido para o domínio das classes da biblioteca.

Indiscutivelmente o paradigma da orientação a objetos apresenta boas perspectivas para a solução de alguns dos problemas confrontados pela Engenharia de Software em relação a reutilização de software. Mas ainda é preciso investir em pesquisa e aguardar a sublimação do mercado das linguagens orientadas a objetos, para que elas se consolidem e ofereçam um risco de investimento menor, e permitam a migração de aplicações de linguagens convencionais. Também, é necessário que sejam criados padrões que permitam o compartilhamento de bibliotecas de classes de objetos entre diferentes linguagens.

2.5 Avaliação e Perspectivas

A reutilização de software progrediu significativamente nos últimos vinte e cinco anos, mas ainda não é uma tecnologia bem definida [PRIE93]. O conceito de reutilização de software apresenta perspectivas reais e concretas para o futuro da Engenharia de Software. O aumento da produtividade, a diminuição de custos de produção e manutenção, a viabilidade de construção de sistemas grandes e complexos com menor esforço são metas que podem ser alcançadas com o amadurecimento desta tecnologia.

O sucesso efetivo da reutilização de software abrange toda uma extensa gama de soluções de problemas, técnicos e não técnicos. São necessárias mudanças no perfil dos produtores de software (programadores, analistas,

projetistas) para que se crie uma cultura de reutilização, pois o esforço dispendido no passado tem que ser aproveitado no futuro.

Para o sucesso da reutilização são necessários planejamento, incentivos, insumos e conscientização. Tracz [TRAC88b] faz uma observação muito importante: "Software reutilizado é diferente de software reutilizável". O que se quer é um processo formal e sistemático de reutilização.

Os benefícios da reutilização variam de um domínio de problemas para outro e de um ambiente para outro. O aumento da produtividade depende das ferramentas, linguagens e métodos empregados [TRAC88b]. São necessárias, portanto, estratégias que definam os melhores caminhos para a reutilização. O potencial das diversas tecnologias de software que viabilizam a reutilização deve ser explorado e avaliadas as possibilidades concretas dessas tecnologias.

É necessário um maior esforço de pesquisa para que se obtenham resultados quantitativos e qualitativos [POUL92] através de experiências práticas, com o objetivo de melhor determinar quais são os fatores que tem influência sobre o processo de reutilização de software.

A reutilização, no final, deve acontecer tão naturalmente que nós nem sequer precisaremos pensar nela [PRIE93].

3 A IDENTIFICAÇÃO DE COMPONENTES REUTILIZÁVEIS: UM ESTUDO DA LITERATURA

3.1 Qualidade de Software

Qualidade só pode ser obtida se puder ser medida, e para ser medida deve ser, antes de tudo, definida. Qualidade não é algo que pode ser agregado ou introduzido a um determinado produto ou processo. Qualidade é um objetivo a ser alcançado e, para tanto, deve estar presente e bem definido desde o início do desenvolvimento do produto.

O conceito de qualidade é expresso através de um conjunto de atributos e características, onde cada um desses atributos e características é responsável por determinado aspecto do conceito de qualidade de software. Portanto, o conceito de qualidade é determinado por um conjunto de atributos e características obtendo-se, assim, a descrição completa da qualidade de um determinado software [ROCH87].

Para obter-se o grau de qualidade desejado para um determinado software, deve-se monitorar todo o processo de desenvolvimento do software, com o objetivo de assegurar o nível de qualidade que se deseja alcançar. Em geral, este processo consta de um ciclo de desenvolvimento que consiste, basicamente, nas fases de definição, projeto, construção e avaliação. Durante estas fases, diversos resultados são obtidos, tais como a especificação de requisitos, a especificação de projeto e, finalmente, os programas. Assim, avaliando a qualidade da especificação de requisitos antes de começar o projeto, e avaliando a qualidade da especificação de projeto antes de começar a construção, garante-se a obtenção do grau de qualidade especificado. Aplicando procedimentos de garantia da qualidade, desde o início do ciclo de desenvolvimento, conseguimos diminuir a possibilidade de que erros sejam

passados para as fases seguintes no processo de desenvolvimento de software.

Determinados atributos e características aplicam-se à maioria dos tipos de software, mas existem outros que só se aplicam, ou são mais relevantes, para tipos específicos de software. O grau desejado de satisfação de um determinado atributo ou característica depende da natureza do software [ARTH85].

É necessário que cada ambiente de desenvolvimento de software determine um conjunto de critérios que permita identificar os atributos mais relevantes do software por ele desenvolvido. Estes atributos são passíveis de quantificação, fornecendo indicadores de comportamento de vários aspectos do software desenvolvido, tais como: número de erros não detectados no software entregue ao usuário, tempo gasto na codificação e teste dos módulos componentes, esforço necessário para manutenção do sistema e outros.

Várias propostas para implementação de programas de garantia da qualidade para ambientes de desenvolvimento de software específicos aparecem na literatura [POOR88], [DASK92], [NUSE93]. São, em geral, propostas de constituição de grupos de trabalho cujo objetivo é definir um conceito próprio de qualidade de software, em termos operacionais, e o estabelecimento de medidas que permitam quantificar este conceito. É importante observar, aqui, que estas medidas não são um fim a ser atingido. A análise dos seus resultados fornece indicadores para o aperfeiçoamento do processo, caracterizando assim uma realimentação do mesmo [DASK92].

Atualmente, o reconhecimento internacional da norma ISO 9000, para software:

> ISO 9001: *Sistemas de Qualidade - Modelo para Garantia da Qualidade em Projetos/Desenvolvimento, Produção, Instalação e Assistência Técnica;*
>
> ISO 9000-3: *Padrões de Gerenciamento da Qualidade e Garantia da Qualidade - Parte 3: Diretivas para Aplicação da ISO 9001 no Desenvolvimento, Suprimento e Manutenção de Software*;

ISO 9126: *Tecnologia da Informação - Avaliação de Produtos de Software - Características de Qualidade e Diretivas para o seu Uso*,

criou um padrão de qualidade que, de forma integral, expressa requisitos gerais para o projeto e desenvolvimento, produção, instalação e manutenção de produtos de software [RIJS93].

Existem na literatura vários modelos para avaliação da qualidade de software, [ARTH85], [BOEH78], [MCCA79], [ROCH87]. O modelo considerado neste trabalho é o modelo de Rocha [ROCH87] para avaliação da qualidade de software. Este modelo será descrito no Capítulo 4, ao tratarmos da nossa proposta para identificação de componentes candidatos à reutilização.

3.2 Identificação de Componentes Reutilizáveis

Livson [LIVS91] afirma que a qualidade é o fator mais importante na reutilização de software, mais ainda do que o aumento da produtividade. Para ele o maior obstáculo para obtenção de qualidade em um software é a "reinvenção da roda" a cada novo projeto.

Existem na literatura várias propostas para implantação de programas de reutilização de código (módulos, componentes) [BURT87], [MATS87], [CALD91], [DUNN91], [DUNN93], [PRIE91a]. Um dos problemas iniciais para se estabelecer um programa de reutilização é a povoação inicial do repositório ou biblioteca de componentes. Segundo Biggerstaff [BIGG87] são necessários altos investimentos de capital, potencial intelectual e tempo antes que a reutilização de código traga um retorno significativo.

O desenvolvimento de componentes reutilizáveis é, em geral, mais caro do que o desenvolvimento de código especializado [LENZ87], devido aos custos adicionais oriundos da manutenção da estrutura do programa (processo) de reutilização - a análise de domínio [PRIE91b], por exemplo.

Um dos pontos chaves para o sucesso do programa de reutilização é que se tenha uma biblioteca de componentes rica e bem organizada. Entretanto, esta biblioteca só estará disponível à organização se o código desenvolvido no passado, sem intenção de reutilização futura, puder ser reutilizado [CALD91].

Uma definição de reutilizabilidade é apresentada em [PETE91] como sendo: *O grau segundo o qual um recurso de software pode ser usado em mais de um programa de computador, ou sistema, ou na construção de outros componentes.* Neste contexto, recurso é definido como *qualquer entidade de software inserida em uma biblioteca com o propósito de ser reutilizada.*

Para extrairmos um componente de código de um sistema já existente, é necessário que ele seja, antes de mais nada, identificado. Para ser identificado, é necessário que ele atenda a uma série de propriedades relacionadas ao seu potencial de reutilização, ou seja, à sua reutilizabilidade. Existe, na literatura, bastante polêmica a respeito da identificação destas propriedades.

Tamanho de um Componente

Biggerstaff e Richter [BIGG87] colocam a questão do tamanho (e benefício) do componente versus potencial de reutilização e observam que, à medida que o tamanho do componente aumenta, o benefício envolvido na reutilização do componente também aumenta. Entretanto, à medida que um componente aumenta de tamanho, ele se torna cada vez mais específico reduzindo, assim, sua generalidade e aumentando o custo de sua reutilização quando são necessárias modificações.

Selby [SELB89] observou que módulos reutilizados sem nenhuma modificação tem um perfil bem definido. Algumas características destes componentes são: tamanho reduzido (número de linhas de código fonte em torno de 150, incluindo linhas de comentários e excluindo linhas em branco), poucos parâmetros de entrada e saída e código bem documentado através de comentários.

Existem na literatura várias propostas para a determinação do tamanho de um componente [CONT86]. A medida clássica, e mais utilizada, é o número de linhas de código fonte. O número de linhas de código fonte é o total de linhas de um componente, não considerando as linhas de comentário e as linhas em branco [CONT86]. Vários trabalhos fornecem indicadores sobre o tamanho, em número de linhas de código fonte, de um módulo [BASI83], [PRIET87a], [CONT86], [SELB89]. Outras medidas de tamanho, propostas por Halstead [HALS77], são descritas mais adiante.

3.2.1 Esforço para Reutilização de um Componente

Para Prieto-Diaz e Freeman [PRIE87a], um componente de software é reutilizável se o esforço necessário para reutilizá-lo é, consideravelmente, menor do que o esforço necessário para implementar um componente com as mesmas funções. Sua proposta para identificação e recuperação de componentes em uma grande biblioteca de componentes é baseada em um esquema de classificação facetado. Neste esquema, as relações genéricas entre os componentes são descritas e classificadas a partir da síntese de suas classes elementares. Medidas de distância conceitual entre os termos do vocabulário usado permitem estabelecer o grau de relacionamento entre componentes. O objetivo é fornecer um ambiente que auxilie na localização e na estimativa do esforço de adaptação do componente, com base na sua adequação à reutilização. Assim sendo, selecionam cinco atributos de qualidade de programa como os indicadores mais relevantes do esforço para reutilização e escolhem uma métrica para cada indicador. A Tabela 3.0 apresenta os atributos e as métricas. Para que estas métricas sejam efetivas estes autores definem os critérios, que julgam ideais, para minimizar o esforço de reutilização. O melhor componente para reutilização (aquele que vai requerer o menor esforço para ser reutilizado) será aquele que satisfizer os quatro primeiros atributos, tendo:

- tamanho pequeno,
- estrutura simples,
- excelente documentação, e

- linguagem de programação igual a do sistema onde será reutilizado.

Atributo	Métrica
Tamanho do programa	Número de linhas de código
Estrutura do programa	Número de módulos, número de ligações e complexidade ciclomática (descrita abaixo)
Documentação do programa	Classificação geral subjetiva (1 a 10)
Linguagem de programação	Proximidade relativa da linguagem
Experiência do (re)usuário	Habilidade em duas áreas: 1) linguagem de programação e 2) domínio da aplicação

Tabela 3.0 - Atributos e métricas de esforço de reutilização [PRIE87a].

A experiência do (re)usuário (explicada abaixo) é um modificador para os três primeiros atributos. Os (re)usuários tem seus próprios critérios, baseados principalmente na experiência, para determinar qual o significado de tamanho pequeno, estrutura simples e excelente documentação. A abordagem usada por Prieto e Freeman é definir um perfil da experiência do (re)usuário e determinar como estes critérios se modificam com o nível de experiência. Como estes critérios não tem valores precisos (são conceitos subjetivos) eles utilizam a teoria de conjuntos fuzzy (conceitos e definições no Capítulo 4) como base para avaliá-los. Por exemplo, para um programador novato, um programa FORTRAN pequeno é algo em torno de quarenta linhas de código fonte, enquanto que, para um programador experiente, é algo em torno de cem linhas de código fonte. Os autores apresentam os conceitos básicos da teoria de conjuntos fuzzy, as curvas das funções fuzzy modificadas segundo a experiência dos (re)usuários e o critério usado para os modificadores fuzzy em função da experiência dos (re)usuários. O trabalho não apresenta os critérios utilizados para geração das curvas, nem detalhes de como foram implementadas.

O que é Complexidade

Para medirmos a complexidade da estrutura lógica de um componente utiliza-se a métrica de complexidade proposta por McCabe [MCCA76]. Esta medida baseia-se no número de caminhos básicos dos comandos de decisão existentes na estrutura de um grafo, considerando-se o programa como um grafo de controle. Existe consenso que o grau de complexidade de um componente tem relação direta com a sua estrutura lógica. McCabe [MCCA76], baseado na teoria dos grafos, propôs uma medida da estrutura lógica, que segundo ele, pode ser obtida pelo número ciclomático. Assim, associa-se a um programa, com pontos únicos de entrada e saída, um grafo. Desta forma, seu número de componentes (p) será 1, cada bloco onde o processamento é sequencial corresponderá a um nó (n) e cada ramo de um desvio corresponderá a uma aresta (e), o valor do número ciclomático de complexidade V(G), deste programa será dado por:

$$V(G) = e - n + 2p$$

Simplificando, este mesmo valor pode ser obtido pelo número de decisões, existentes no módulo, mais um:

$$V(G) = IF's + 1$$

Um refinamento desta medida propõe que se considere o número total de condições, já que uma estrutura do tipo IF (c1 AND c2) THEN... é na realidade equivalente a IF (c1) THEN IF (c2) THEN... . Desta forma, teríamos o número ciclomático Vm(G) igual a:

$$Vm(G) = IF's + AND's + OR's + NOT's + 1$$

McCabe sugere como limite superior para sua métrica o valor 10.

3.2.2 Determinando o Potencial de Reutilização

Dunn e Knight [DUNN91] fazem uma seleção manual de componentes através de sistemas já existentes, utilizando uma série de critérios informais obtidos através de reuniões mantidas com os desenvolvedores dos sistemas pesquisados. Não fazem, entretanto, distinção entre forma e função (dos componentes) no seu processo de seleção, apesar de inicialmente procurarem padrões. Acabam por identificar categorias funcionais específicas, nas quais os componentes reutilizáveis poderiam ser agrupados. Em [DUNN93], é apresentada uma ferramenta baseada em um sistema especialista para pesquisa, em sistemas de software existentes, de componentes de código reutilizáveis. Utilizam uma base de conhecimentos criada a partir de informações, de todas as fases do processo de desenvolvimento, obtidas junto aos desenvolvedores dos sistemas pesquisados [DUNN91]. Segundo o modelo proposto, só interessam componentes com grande potencial de reutilização que são identificados através de três critérios:

- contagem do número de referências a uma função ou procedimento, identificando os mais frequentemente referenciados como potencialmente reutilizáveis,

- identificação de módulos que apresentem um fraco acoplamento com outros módulos, e,

- identificação de módulos com alta coesão.

Yglesias [YGLE93] estabelece um paralelismo entre reutilização de software e reutilização de informações, enumerando as características necessárias para um componente de software ser classificado como reutilizável:

- Facilidade de compreensão - O componente deve ser completamente documentado, incluindo código bem documentado através de linhas de comentário.

- Completeza funcional - O componente tem todas as operações necessárias para atender aos requisitos atuais e quaisquer requisitos futuros razoáveis dentro do contexto.

- Uniformidade - O componente usa notação, estruturas de controle e chamadas consistentes e relaciona logicamente os mesmos objetos em qualquer nível.

- Modularidade - O componente tem boa estrutura para organizar dados e algoritmos. O componente também apresenta as propriedades desejáveis de fraco acoplamento (poucas interdependências) e forte coesão (o componente executa uma função específica e bem definida).

- Confiabilidade - O componente executa de forma consistente a função anunciada sem erros. O componente foi testado repetidamente em vários sistemas operacionais e diferentes tipos de hardware.

- Bom tratamento de erros e exceções - O componente isola, documenta e trata erros de forma consistente. Contém uma variedade de opções para responder aos possíveis erros.

- Portabilidade - O componente não depende dos serviços de um único hardware ou sistema operacional.

Weide, Ogden e Zweben [WEID91] definem o conceito de componente reutilizável abstrato que, segundo eles, deve:

- ser claro e compreensível (clareza),

- expressar tudo sobre o comportamento esperado de uma implementação correta e nada mais (restrito),

- suportar uma variedade de implementações (generalidade),

- exportar operações cuja funcionalidade é tão básica que não pode ser obtida através da combinação da exportação de outras operações, ou seja, deve exportar operações primárias (primitivo),

- exportar operações primárias que em conjunto podem oferecer funcionalidade suficiente para que um cliente execute uma ampla classe de computações de interesse com o componente (suficiência),

- não depender do comportamento de outro componente abstrato para explanação da sua funcionalidade (fraco acoplamento),

- encapsular um conceito simples que não possa ser decomposto (alta coesão).

Isoda [ISOD92] apresenta uma lista de atributos para validação de componentes reutilizáveis, com uma série de requisitos mandatórios e recomendações. Alguns itens (mandatórios) desta lista de validação são:

- satisfazer os princípios de boa estruturação (estruturabilidade),

- atender aos padrões do ambiente de desenvolvimento de software (padronização),

- possuir alta confiabilidade, e,

- ser genérico (generalidade).

Segundo Basili [BASI90] a qualidade de um objeto (componente) reutilizável depende da sua: maturidade (número de sistemas que o utilizam), complexidade (complexidade ciclomática) e confiabilidade (número de falhas durante suas utilizações anteriores).

Hislop [HISL91] apresenta um interessante atributo relacionado à reutilizabilidade de um componente: vida útil. Observa que, à medida que a tecnologia evolui e as organizações se transformam, a reutilizabilidade de um

componente tende a diminuir. Observa, ainda, que tanto a forma como a função de um componente tem impacto substancial no potencial de reutilização. Conclui que as métricas atuais são úteis para identificação de certas características em módulos reutilizados, mas não são boas discriminadoras de semelhanças e diferenças entre módulos. Módulos funcionalmente similares podem ter valores métricos muito diferentes e módulos funcionalmente diferentes podem apresentar valores métricos similares.

3.2.3 Determinando a Qualidade de um Componente

Caldiera e Basili [CALD91] identificam três atributos básicos para reutilização: custo, utilidade funcional no contexto do domínio da aplicação e qualidade. A cada um desses atributos são associadas métricas para medir o atributo diretamente, ou mesmo indiretamente, através de evidências da sua existência. Os três atributos são assim definidos:

Custo - os custos de reutilização consistem do custo de extração do componente de sistemas antigos, custo de empacotamento na forma de um componente reutilizável, e custo para encontrar, modificar e integrar um componente em um novo sistema. São, portanto, desejados componentes pequenos e simples, e usadas métricas de volume e complexidade (descritas abaixo) como indicadores. Estas métricas permitem, também, avaliar indiretamente a legibilidade, estruturabilidade (observação dos princípios de boa estruturação) e não-redundância (não repetição de código) de implementação de um componente.

Utilidade funcional - é afetada pela comunidade (no sentido de qualidade do que é comum) e variedade de funções desempenhadas por um componente. Estes atributos são avaliados através das métricas de volume e regularidade. A regularidade pode ser uma medida indireta da utilidade funcional de um componente se for considerado o número de vezes que o componente ocorre dentro do sistema analisado (assumindo-se que um componente frequentemente chamado, ou ativado, apresenta um alto potencial de reutilização).

Qualidade - Vários atributos de qualidade são importantes para a reutilização de componentes: correção, legibilidade, testabilidade, facilidade de modificação e desempenho. Para automatização do processo de identificação de candidatos à reutilização, são consideradas medidas indiretas de tamanho e legibilidade como indicadores da correção, e o número de caminhos independentes como medida da testabilidade. Os atributos correção e testabilidade são avaliados através de medidas de volume e complexidade ciclomática. Facilidade de modificação é refletida na medida da legibilidade do componente.

As métricas utilizadas por Caldiera e Basili [CALD91] são: Regularidade (Halstead [HALS77]), Volume (Halstead [HALS77]), Complexidade Ciclomática (McCabe [MCCA76]), e Frequência de Reutilização, que descrevemos a seguir.

Frequência de Reutilização

Vamos considerar como componente desenvolvido um componente (rotina, função) que foi desenvolvido por programadores em um ambiente de desenvolvimento de software. E componente padrão uma função básica de determinada linguagem de programação (como o printf da linguagem C, por exemplo). A métrica **frequência de reutilização** parte do princípio de que ao se comparar o número de chamadas a um componente desenvolvido com o número de chamadas efetuadas a uma classe de componentes padrão, assumidamente reutilizáveis, podemos associar um valor de frequência de reutilização a um componente desenvolvido. O objetivo é identificar quantas vezes um determinado componente desenvolvido é chamado e estabelecer uma relação com uma função padrão reconhecidamente reutilizável. Assim, é associado a cada componente identificado, uma medida estática da sua frequência de reutilização (*fr*) em relação ao sistema ao qual pertence. A razão entre o número (*n*) de chamadas feitas a um componente desenvolvido (*C*) e o número médio (*m*) de chamadas feitas a um componente padrão (*P*) determina a frequência de reutilização do componente desenvolvido. Portanto, a frequência de reutilização (*fr*) de um componente desenvolvido é dada por:

$$fr = n(C) \;/\; m(P).$$

Métricas de Halstead

Halstead [HALS77] baseado em aspectos da teoria de informação e da psicologia cognitiva e apoiado em resultados empíricos significativos, concluiu que a implementação de um algoritmo em qualquer linguagem apresenta características próprias. Assim sendo, existe um conjunto de operadores e operandos que medidos e agregados de forma adequada permitem caracterizar o seu tamanho, esforço para desenvolvê-lo e até estimar o número de erros que aquele componente irá conter quando pronto.

Tamanho

A medida de tamanho (número de ocorrência de operadores e operandos) consiste na contagem de operadores e operandos. Na métrica de tamanho proposta por Halstead [HALS77] são contados os operadores (n_1) e operandos (n_2) e o número de ocorrências de cada um deles. O total do número de ocorrência de operadores (N_1) e operandos (N_2) corresponde ao tamanho (N) do componente.

n_1	- número de operadores distintos
n_2	- número de operandos distintos
N_1	- número total de operadores
N_2	- número total de operandos
$N = N_1 + N_2$	- tamanho

Regularidade

Pode-se medir a utilização de práticas de programação corretas através do quanto conseguimos estimar o tamanho de um componente com base em um critério de regularidade. Como proposto por Caldiera e Basili [CALD91] utiliza-se, novamente, a teoria de Halstead [HALS77] onde temos o tamanho (N) real de um componente

$$N = N_1 + N_2$$

e o tamanho estimado ($N^{\wedge}$)

$$N^{\wedge} = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

A proximidade do valor estimado é uma medida de regularidade (r) da codificação do componente:

$$r = N^{\wedge} / N$$

A regularidade de um componente mede a legibilidade e a não-redundância na sua implementação. Assim, valores de regularidade nas proximidades de 1 são desejáveis.

Volume

Na métrica de volume proposta por Halstead [HALS77] são contados os operadores (n_1) e operandos (n_2) e o número de ocorrências de cada um deles. O total do número de ocorrência de operadores (N_1) e operandos (N_2) corresponde ao tamanho (N) do componente. O produto do tamanho (N) vezes o logaritmo na base 2 do vocabulário (n) corresponde ao volume do componente.

$n = n_1 + n_2$ - vocabulário

$V = N \times \log_2 n$ - volume

No modelo de qualidade proposto por Arthur [ARTH85] o fator reutilizabilidade é avaliado através de métricas de software. Deve-se observar, entretanto, que o peso de cada métrica é função do software e de considerações do ambiente em questão. Um componente é avaliado de acordo com os seguintes critérios:

- a amplitude de seu potencial de aplicação (generalidade),
- o grau de independência do componente de software com relação ao hardware sob o qual é executado (independência de hardware),

- a independência funcional dos componentes de software de um programa (modularidade),

- índice segundo o qual o código fonte do componente fornece informações significativas (auto-documentação), e,

- o grau independência do componente de software com relação a aspectos não padronizados da linguagem de programação, sistema operacional e outras restrições ambientais (independência do sistema de software).

3.3 Processos de Reutilização

Prieto-Diaz [PRIE91a] apresenta um modelo para implementação de um programa de reutilização de software, fundamentado em uma estratégia incremental. Neste trabalho, são discutidos aspectos técnicos e aspectos não-técnicos de ordem gerencial, econômica, cultural, de desempenho e de transferência de tecnologia. Sua proposta consiste no estabelecimento de um programa de reutilização de software incremental, sistemático e formal para organizações de software. Incremental significa que o programa é implementado em estágios sucessivos, onde cada estágio estabelece as bases para o estágio seguinte. Sistemático significa que o processo é consistente e repetitivo, seguindo uma sequência lógica de eventos. Por fim, formal, neste modelo, significa que o processo pode ser decomposto em etapas bem definidas. Antes da implantação do programa deve ser gerado um relatório de avaliação com os seguintes estudos: análise de exequibilidade; adequabilidade do domínio; análise de custo/benefício e plano de implementação. O programa de reutilização proposto pode ser implementado em quatro estágios básicos: 1) Inicial, 2) Expansão, 3) Contração, e 4) Estado Estacionário. A Figura 3.0 apresenta o modelo proposto. No estágio inicial (1), o software existente é analisado para selecionar componentes com potencial de reutilização. Descritores dos componentes são extraídos de forma manual, ou automaticamente, e é gerado um índice preliminar. No estágio de expansão (2), o tamanho do catálogo de componentes aumenta à medida que mais

software existente é identificado como reutilizável. Isto indica a necessidade de um esquema de classificação.

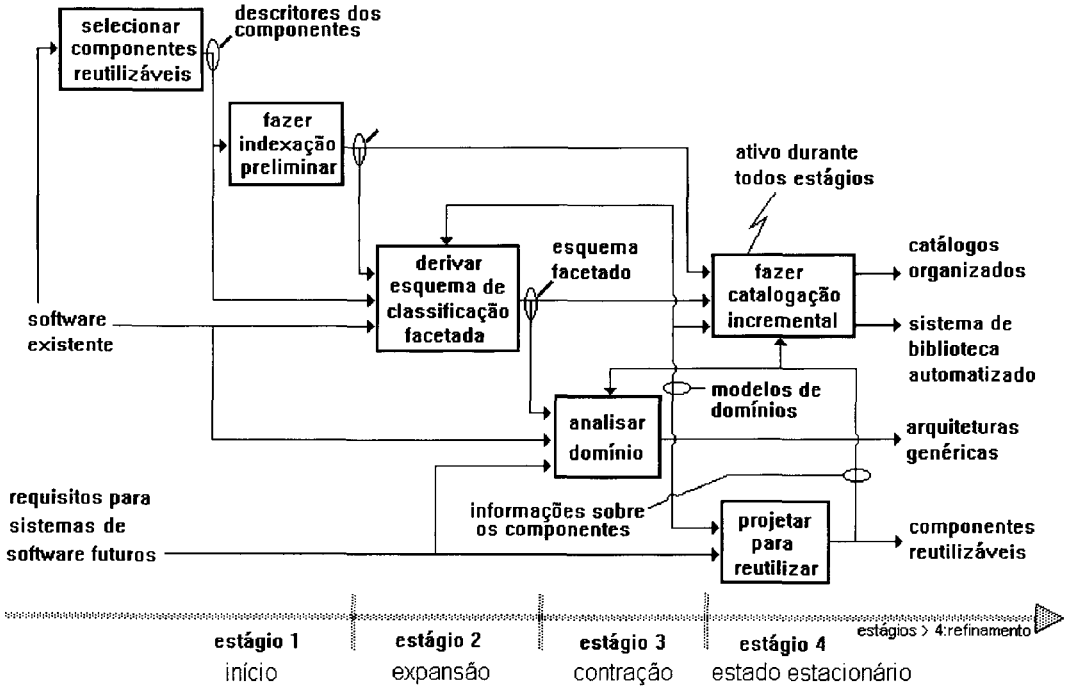


Figura 3.0 - Implementação de um programa de reutilização [PRIE91a].

Um esquema de classificação facetada inicial é gerado, e o resultado da classificação é incluído ao catálogo do estágio 2. Este esquema facetado fornece modelos básicos do domínio na forma de taxonomias e descritores padrão, que por sua vez dão partida ao processo de análise de domínio. O estágio de contração (3) é responsável pela atividade chave - a análise de domínio. Dados do segundo estágio, em conjunto com informações obtidas do software existente e dos requisitos para futuros sistemas, são usados para a análise de domínio. Arquiteturas padrão e modelos funcionais são derivados e componentes comuns são agrupados para suportar funções genéricas básicas. Isto resulta numa contração do tamanho do catálogo em função da sua atualização, e também da classificação, gerando o novo catálogo do estágio 3. No estágio de estado estacionário (4), após a identificação dos elementos essenciais para os sistemas de software de um domínio específico, os componentes existentes são substituídos progressivamente por componentes que suportam funções específicas do domínio. Outros estágios

do programa não devem aumentar o tamanho da coleção de componentes, devem somente tornar os componentes atuais mais eficientes e confiáveis.

Caldiera e Basili [CALD91] apresentam um modelo para implementação da reutilização no estilo de uma fábrica de software. O modelo proposto divide o modelo tradicional de ciclo de vida em duas partes: a primeira, o projeto, entrega sistemas de software, enquanto que a segunda, a fábrica, fornece objetos de software reutilizáveis para o projeto. O objetivo primário do modelo é a identificação, extração e empacotamento de componentes reutilizáveis, trabalhando com um conhecimento específico do domínio de aplicação do qual um componente é extraído. Sua proposta consiste numa estratégia para extração de componentes, a partir de sistemas já existentes, dividida em duas fases (Figura 3.1).

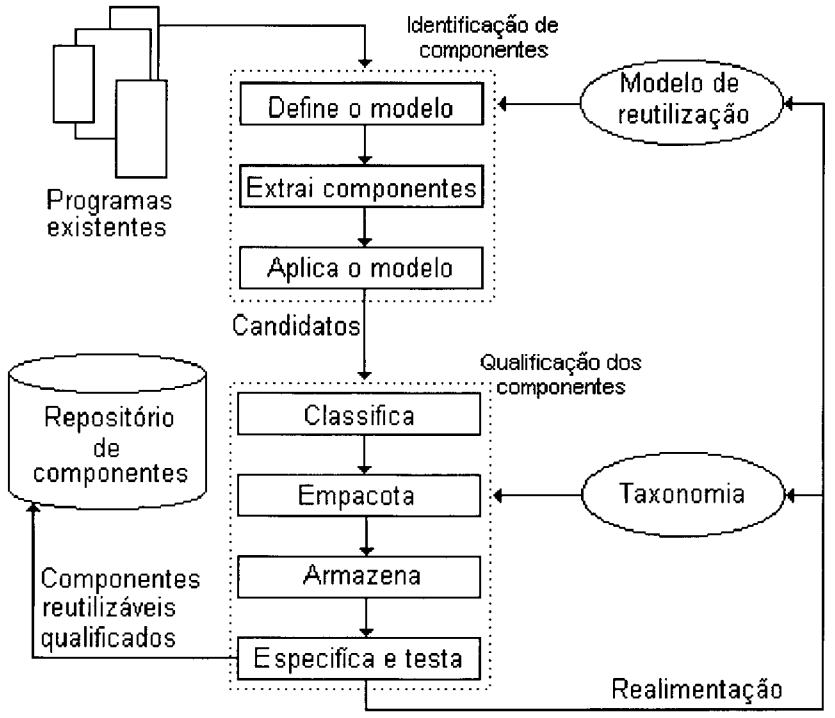


Figura 3.1 - Extração de componentes [CALD91].

Na primeira fase, que é completamente automatizada, são identificados os componentes candidatos à reutilização, de acordo com um modelo de atributos básicos de reutilizabilidade. Na segunda fase um especialista do domínio de

aplicação analisa os candidatos selecionados, avaliando suas possibilidades de reutilização. A divisão em duas fases tem por objetivo diminuir a quantidade de intervenção humana no processo o que consequentemente diminui o seu custo.

Na primeira fase, identificação de componentes reutilizáveis, são definidas três etapas: 1) definição (ou refinamento) do modelo de atributos de reutilizabilidade; 2) extração de componentes, e, 3) aplicação do modelo. Na primeira etapa, através da sua interpretação particular acerca dos atributos de um componente potencialmente reutilizável no ambiente, definem um conjunto de métricas que quantificam estes atributos e, estabelecem um intervalo aceitável de valores para as métricas escolhidas. Os resultados, obtidos através da aplicação do modelo, permitem que os valores de aceitação sejam continuamente atualizados no sentido de obter um modelo de atributos de reutilizabilidade que maximize as chances de selecionar componentes candidatos à reutilização. Na segunda etapa, são extraídas unidades modulares de sistemas existentes para serem completadas com quaisquer referências externas (blocos de código comuns a vários programas, por exemplo), necessárias para serem reutilizadas independentemente. Por unidade modular entenda-se uma função C ou uma subrotina FORTRAN. Na terceira etapa, o modelo de atributos de reutilizabilidade é aplicado nos componentes completos extraídos. Os componentes, cujas medidas estão dentro dos intervalos de valores aceitáveis, tornam-se candidatos a componentes reutilizáveis a serem analisados pelo especialista do domínio na fase de qualificação.

Na segunda fase, qualificação dos componentes extraídos, o especialista do domínio: 1) gera uma especificação funcional do componente, 2) testa o componente através de uma série de casos de teste, 3) classifica o componente segundo um conjunto de atributos identificados na análise de domínio, 4) cria o manual do (re)usuário com uma série de informações sobre o componente, 5) armazena o componente no repositório junto com as informações e dados gerados até o momento sobre o componente, e 6) atualiza os atributos de reutilizabilidade em função das observações feitas até aqui.

No Capítulo 4, partindo-se da estrutura do modelo de avaliação da qualidade de programas, de trabalhos desenvolvidos na COPPE e da revisão da literatura (Capítulo 3), definimos os atributos que devem ser considerados para a avaliação do potencial de reutilização de componentes de código. Trataremos da área de software científico, e de engenharia, que utilizam a linguagem FORTRAN. É, também, desenvolvida uma ferramenta de apoio à seleção de componentes de código candidatos à reutilização que utiliza métricas de software e conceitos da teoria de conjuntos fuzzy, através do uso de variáveis e termos linguísticos.

4 AVALIAÇÃO DA REUTILIZABILIDADE DE COMPONENTES

4.1 Avaliação da Reutilizabilidade de Componentes Segundo o Modelo de Rocha

Rocha [ROCH83] propôs um modelo para avaliação da qualidade de software baseado nos seguintes conceitos :

1. **Objetivos de qualidade:**
   - são as propriedades gerais, que o produto deve possuir;

2. **Fatores de qualidade:**
   - determinam a qualidade na visão dos diferentes usuários do produto - usuário final e outros;

3. **Critérios:**
   - são atributos primitivos, possíveis de serem avaliados;

4. **Processo de avaliação:**
   - determinam as métricas a serem utilizadas, de forma a se medir o grau de presença, no produto, de um determinado critério;

5. **Medidas:**
   - são o resultado da avaliação do produto, segundo os critérios;

6. **Medidas agregadas:**
   - são o resultado da agregação das medidas obtidas ao se avaliar de acordo com os critérios, e quantificam os fatores.

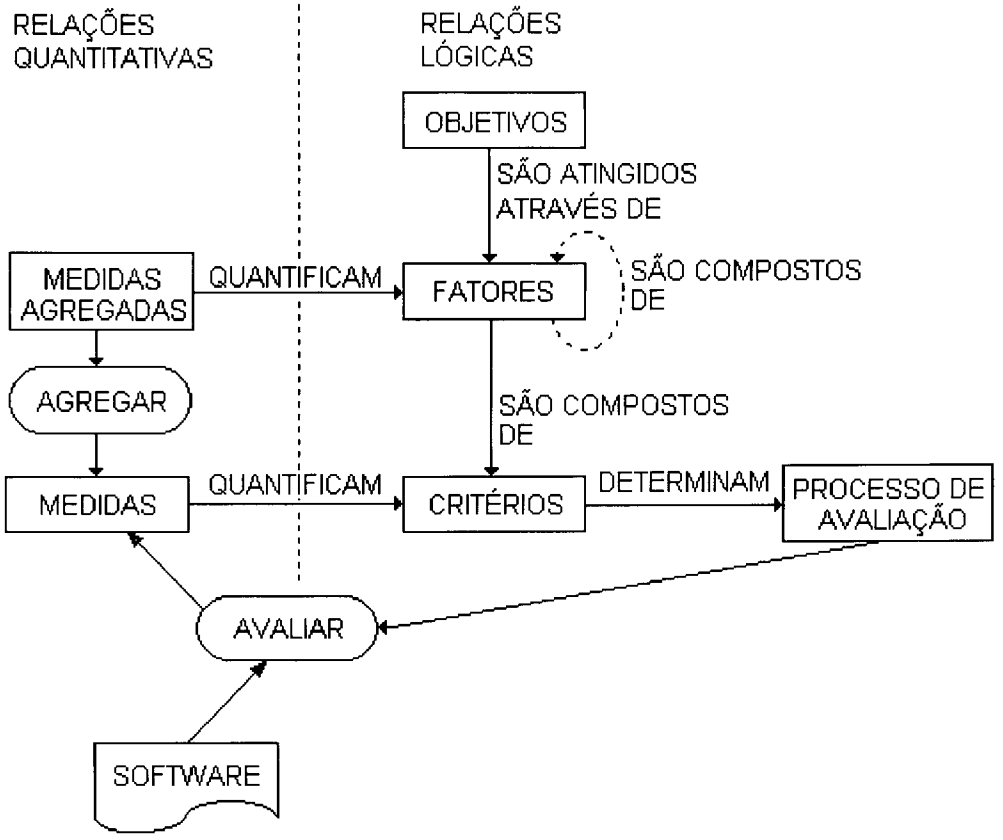


Figura 4.0 - Estrutura do método proposto por Rocha [ROCH87] para avaliação da qualidade de software.

Neste trabalho, tratamos exclusivamente da avaliação da qualidade de programas, pois estamos interessados na identificação de atributos de reutilizabilidade em componentes de código (programas, subrotinas, funções, etc.). Rocha define três objetivos para avaliação da qualidade de programas: *Confiabilidade da Representação, Confiabilidade Conceitual* e *Utilizabilidade*. A Tabela 4.0 mostra os objetivos, fatores e subfatores de qualidade de programas.

	Objetivos	Fatores	Subfatores
Qualidade de Programas	Confiabilidade da Representação	Legibilidade	Clareza
			Concisão
			Estilo
			Modularidade
		Manipulabilidade	Disponibilidade
			Estrutura
			Rastreabilidade
	Utilizabilidade	Avaliabilidade	Verificabilidade
			Validabilidade
		Eficiência	
		Manutenibilidade	
		Operacionalidade	Amenidade ao uso
			Oportunidade
		Portatilidade	
		Rentabilidade	
		Reutilizabilidade	
	Confiabilidade Conceitual	Fidedignidade	Completeza
			Necessidade
			Precisão
		Integridade	Robustez
			Segurança

Tabela 4.0 - Objetivos, fatores e subfatores para avaliação da qualidade de programas [ROCH87].

Segundo Rocha [ROCH87] os objetivos de qualidade são, assim, definidos:

- **Confiabilidade da Representação** refere-se às características de representação do programa que afetam a sua compreensão e manipulação por pessoas. Este objetivo é atingido através dos fatores *legibilidade* e *manipulabilidade*.

- **Confiabilidade Conceitual** é um objetivo essencial de ser atingido para que o programa atinja às necessidades e requisitos que motivaram sua construção. A confiabilidade conceitual é alcançada através dos fatores *fidedignidade* e *integridade*.

- **Utilizabilidade** exige tanto a confiabilidade conceitual quanto a confiabilidade da representação e determina a conveniência e a viabilidade da utilização do programa ao longo de sua vida útil. A utilizabilidade é alcançada através dos fatores *avaliabilidade*, *eficiência*, *manutenibilidade*, *operacionalidade*, *portatilidade*, *rentabilidade* e ***reutilizabilidade***.

Este modelo vem sendo utilizado para a avaliação dos produtos gerados em todas as fases de desenvolvimento de software: especificações, projetos e programas. Para isto foram definidos atributos e métricas, relativos a estes produtos, bem como implementadas ferramentas de apoio à avaliação. Entre os trabalhos realizados, nesta linha, destacam-se os trabalhos de Andrade [ANDR91] e Belchior [BELC92], na avaliação da qualidade de programas, que detalhamos a seguir.

Para que o processo de controle da qualidade seja, de fato, vantajoso, viável e utilizável, é preciso que se usem ferramentas automatizadas. A coleta manual de dados, durante a avaliação, é um processo tedioso, propenso a erros, além de demandar muito tempo. Portanto, ferramentas automatizadas implicam na melhoria da coleta e da análise dos dados, oriundos da aplicação das métricas, na redução dos custos desse processo e na disponibilidade mais rápida dos resultados sempre que for necessário [ANDR91]. Com este objetivo Andrade desenvolveu uma ferramenta de análise estática para apoiar a avaliação da qualidade de programas FORTRAN. Esta ferramenta foi desenvolvida a partir da determinação de uma série de critérios e processos de avaliação para os fatores e subfatores relacionados ao objetivo Confiabilidade da Representação.

A Tabela 4.1 relaciona os subfatores, critérios e suas descrições para o fator Legibilidade do objetivo Confiabilidade da Representação para avaliação da qualidade de programas.

Subfatores	Critérios	Descrições
Clareza	Número de Decisões	O programa apresenta uma quantidade máxima de comandos de decisão segundo a medida de complexidade ciclomática de McCabe.
	Padronização	O programa obedece aos padrões técnicos de programação estabelecidos tanto pelo FORTRAN 77 ANSI como pela instituição a que pertence.
Concisão	Não Anomalia	O programa não implementa práticas desnecessárias ou procedimentos não utilizados.
	Não Repetição	O programa não apresenta uma mesma sequência de código repetida em diferentes lugares.
Estilo	Indentação	O programa utiliza afastamentos da margem esquerda para uma boa apresentação da sua estrutura lógica, destaca níveis de aninhamento e blocos de comandos.
	Comentário	O programa contém comentários com explicações sobre a sua lógica e funcionamento.
	Identificação	O programa utiliza nomes significativos, apropriados e mnemônicos.
	Organização Visual	O programa apresenta nomes e comandos dispostos de forma organizada.
	Programação Estruturada	O programa obedece às normas da técnica de programação estruturada.
Modularidade	Balanceamento	Apenas as subrotinas de mais baixo nível, do programa, tratam de características físicas de implementação.
	Coesão	O programa apresenta um forte relacionamento entre os argumentos dos seus subprogramas.
	Não Acoplamento	O programa possui um fraco interrelacionamento entre as suas subrotinas.
	Não Memorização	O programa não utiliza qualquer tipo de memória para manter o estado das variáveis entre ativações sucessivas de suas subrotinas.
	Número de Módulos Inferiores	O programa apresenta, para cada unidade de programa, um baixo número de subprogramas respectivamente chamados pelas mesmas.
	Número de Módulos Superiores	O programa apresenta, para cada um de seus subprogramas chamados, um número adequado de unidades de programa que respectivamente os chamem.
	Tamanho	As unidades do programa não ultrapassam um número padrão de linhas de código fonte executáveis.

Tabela 4.1 - Subfatores, critérios e suas descrições, do fator Legibilidade relacionados ao objetivo Confiabilidade da Representação para avaliação da qualidade de programas [ANDR91].

Através dos resultados de uma pesquisa de campo realizada em várias instituições financeiras, Belchior [BELC92] identificou os principais atributos de qualidade de um software financeiro e traçou o perfil de qualidade deste tipo de produto de software. Neste trabalho foram estabelecidos vários subfatores

e critérios para o modelo de qualidade de software proposto por Rocha. Nosso interesse sobre este trabalho recai sobre os três subfatores: **Generalidade**, **Aplicabilidade** e **Adaptabilidade**, definidos para o fator Reutilizabilidade, relacionado ao objetivo Utilizabilidade, para avaliação da qualidade de programas. Estes subfatores, suas respectivas definições, seus critérios e descrições são apresentados na Tabela 4.2.

Subfatores e suas definições	Critérios	Descrições
Generalidade Refere-se à característica que um programa deve ter, para que seus módulos tenham o mínimo de restrições de tipos e de quantidade de dados.	Independência do Tipo de Dados	Aptidão de um programa operar com os vários tipos de dados da linguagem utilizada.
	Independência da Quantidade de Dados	Característica de um programa não possuir restrições, para utilização de qualquer volume de dados.
Aplicabilidade É a característica de um programa poder ser reutilizado ao se especificar outra aplicação, na mesma área do domínio do problema.	Identificação dos Componentes	Consiste na padronização dos nomes dados aos componentes de software existentes, com o objetivo de facilitar a sua reutilização.
	Recuperação dos Componentes	É a facilidade de se recuperar componentes de software existentes.
Adaptabilidade É a habilidade em se trocar, facilmente, componentes de código, envolvendo-os com as modificações requisitadas pelo meio.	Composição Tecnológica	A composição dos componentes de código deve ser de tal maneira que, em geral, não necessitem ser modificados, para serem reutilizados.

Tabela 4.2 - Subfatores, suas definições, critérios e descrições, do fator Reutilizabilidade relacionado ao objetivo Utilizabilidade para avaliação da qualidade de programas [BELC92].

4.2 O Fator Reutilizabilidade

A partir da estrutura do modelo proposto por Rocha [ROCH87], dos trabalhos de Andrade [ANDR91], Belchior [BELC92] e da revisão da literatura realizada (Capítulo 3) identificamos os seguintes subfatores e critérios para avaliação da reutilizabilidade de código (Tabelas 4.3, 4.4, 4.5, 4.6 e 4.7).

Fator	Subfator	Critérios	Descrições
Reutilizabilidade	**Generalidade**	**Independência do Tipo de Dados**	A aptidão de um programa operar com vários tipos de dados da linguagem utilizada.
		Independência da Quantidade de Dados	Característica de um programa não possuir restrições, para utilização de qualquer volume de dados.
		Independência de Compilador	A característica da codificação de um componente não incluir particularidades de um determinado compilador.
		Independência de Hardware	É o grau de independência do componente de software em relação ao hardware para o qual foi, originalmente,desenvolvido.

Tabela 4.3 - Subfator Generalidade do fator Reutilizabilidade relacionado ao objetivo Utilizabilidade. É avaliado através dos critérios Independência do Tipo de Dados, Independência da Quantidade de Dados, Independência de Compilador e Independência de Hardware.

Fator	Subfator	Critérios	Descrições
Reutilizabilidade	**Maturidade**	**Confiabilidade**	A probabilidade de um componente executar satisfatoriamente sua função (sem falhas) durante um período de tempo.
		Número de Utilizações	Medida do número de vezes que o componente foi (re)utilizado.
		Vida Útil	É o período de tempo entre cada (re)utilização do componente.

Tabela 4.4 - Subfator Maturidade do fator Reutilizabilidade, relacionado ao objetivo Utilizabilidade. É avaliado através dos critérios Confiabilidade, Número de Utilizações e Vida Útil.

Fator	Subfator	Critérios	Descrições
Reutilizabilidade	**Simplicidade**	**Complexidade**	Número de caminhos lógicos em um programa ou componente.
		Regularidade	É a razão entre o tamanho estimado e o tamanho real de um componente.
		Tamanho	Os componentes tem sua extensão compreendida entre valores mínimo e máximo, estabelecidos como padrão.

Tabela 4.5 - Subfator Simplicidade do fator Reutilizabilidade, relacionado ao objetivo Utilizabilidade. É avaliado através dos critérios Complexidade, Regularidade e Tamanho.

Fator	Subfator	Critérios	Descrições
Reutilizabilidade	**Estilo**	**Documentação Interna**	Refere-se a característica do código fonte do componente apresentar informações significativas através de comentários.
		Organização Visual	É a característica de um componente ter uma boa apresentação quanto ao posicionamento de nomes, comandos, comentários, linhas em branco e na constatação de que foram utilizadas boas práticas de programação na sua implementação.
		Padronização	O componente obedece às normas e padrões estabelecidos pelo ambiente de programação da organização.
		Programação Estruturada	O componente obedece às normas da técnica de programação estruturada.

Tabela 4.6 - Subfator Estilo do fator Reutilizabilidade, relacionado ao objetivo Utilizabilidade. É avaliado através dos critérios Documentação Interna, Organização Visual, Padronização e Programação Estruturada.

Fator	Subfator	Critérios	Descrições
Reutilizabilidade	**Modularidade**	**Fan-out**	Número de módulos inferiores (número de módulos chamados).
		Fan-in	Número de módulos superiores (número de módulos chamantes).
		Acoplamento	É a relação de interdependência entre componentes.
		Não Memorização	O componente não possui memória de existência prévia, executando, a cada ativação, como se fosse a primeira vez, ou seja, sem memória de estados anteriores.

Tabela 4.7 - Subfator Modularidade do fator Reutilizabilidade, relacionado ao objetivo Utilizabilidade. É avaliado através dos critérios Fan-out, Fan-in, Acoplamento e Não Memorização.

Atributos de qualidade são, na sua maioria, conceitos subjetivos e de difícil avaliação. Portanto, o uso de uma teoria que trate dessa subjetividade de forma adequada adapta-se bem às necessidades de medição desses atributos. Assim, o uso da teoria de conjuntos fuzzy aliada ao uso de variáveis e termos linguísticos oferece boas perspectivas de uso neste trabalho.

4.3 Lógica Fuzzy

A lógica fuzzy é uma lógica de valores múltiplos que define níveis ou graus de pertinência de um elemento em um conjunto - uma forma prática de lidar com questões do mundo real. Segundo Zadeh [ZADE73], [ZADE84], o criador da lógica fuzzy, ela é *um tipo de lógica que utiliza graduações ou declarações qualificadas, ao invés daquelas que são estritamente verdadeiro ou falso.*

Um conjunto fuzzy é *um conjunto que não tem um grau de pertinência rígido, ou seja, permite que os objetos tenham graus ou níveis de pertinência no intervalo unitário [0,1]* [ZADE84]. Bezdek [BEZD93] observa que os conjuntos fuzzy são uma generalização da teoria convencional de conjuntos que permitem representar a imprecisão do cotidiano.

Outro conceito importante é o de variável linguística, que segundo Zadeh [ZADE84] *são termos ordinários de linguagem que são usados para representar um conjunto fuzzy particular em um dado problema, tais como "grande", "pequeno", "médio" ou "OK".*

Modificadores fuzzy segundo Zadeh [ZADE84] são *operações que alteram a função de pertinência de um conjunto fuzzy, através da expansão da região de transição entre o grau de pertinência máximo e o de não-pertinência, através da mudança de forma da região de transição, ou através do deslocamento da região de transição.*

Conjuntos convencionais contêm objetos que satisfazem propriedades específicas necessárias para serem membros do conjunto. O conjunto de números *H,* Figura 4.4 (a), de 6 a 8 é convencional e descrito por

$$H = \{ r \in \Re \mid 6 \leq r \leq 8 \}$$

De forma equivalente, *H* é descrito pela sua *função de pertinência* (FP),

$$p_H : \Re \rightarrow \{0,1\}$$

,definida como

$$p_H(r) = \left\{ \begin{array}{l} 1;\ 6 \leq r \leq 8 \\ 0;\ r < 6 \text{ ou } r > 8 \end{array} \right\}$$

O conjunto convencional *H* e o gráfico de p_H são mostrados na Figura 4.4 (a) e (c) respectivamente. Enquanto que p_H mapeia todos os números reais *r* no intervalo fechado [0,1], os conjuntos convencionais correspondem a uma lógica binária: é ou não é, 1 ou 0, pertence ou não pertence [BEZD93].

Consideremos agora o conjunto *F* dos números reais *próximos a 7*. Uma vez que a propriedade "próximos a 7" é imprecisa (fuzzy), não existe somente uma função de pertinência para *F*. Pelo contrário, o modelador deve decidir, baseado na aplicação e nas propriedades desejadas para *F*, como ela deve ser. Propriedades que podem ser consideradas para *F* são:

1. Normalidade ($p_F(7) = 1$),

2. Monotonicidade (quanto mais próximo de 7 estiver *r*, mais próximo de 1 estará $p_F(r)$, e vice-versa), e

3. Simetria (números igualmente distantes à direita e à esquerda de 7 devem ter o mesmo grau de pertinência).

As funções apresentadas na Figura 4.4 (b) e (d) podem ser representações úteis de *F*. p_{F1} é discreta (o gráfico escada), enquanto que p_{F2} é contínua (o gráfico triangular). Pode-se facilmente construir uma FP para *F*, de forma que todo número tenha um grau de pertinência positivo em *F*, por exemplo.

Uma das grandes diferenças entre os conjuntos convencional e fuzzy, é que o primeiro tem sempre uma única função de pertinência, enquanto que todo conjunto fuzzy tem um número infinito de funções de pertinência que podem representá-lo. Isto representa, ao mesmo tempo, fragilidade e força; a unicidade é sacrificada, mas oferece um ganho paralelo em termos de flexibilidade, permitindo que os modelos fuzzy possam ser "ajustados" para se obter um aproveitamento máximo em uma dada situação [BEZD93].

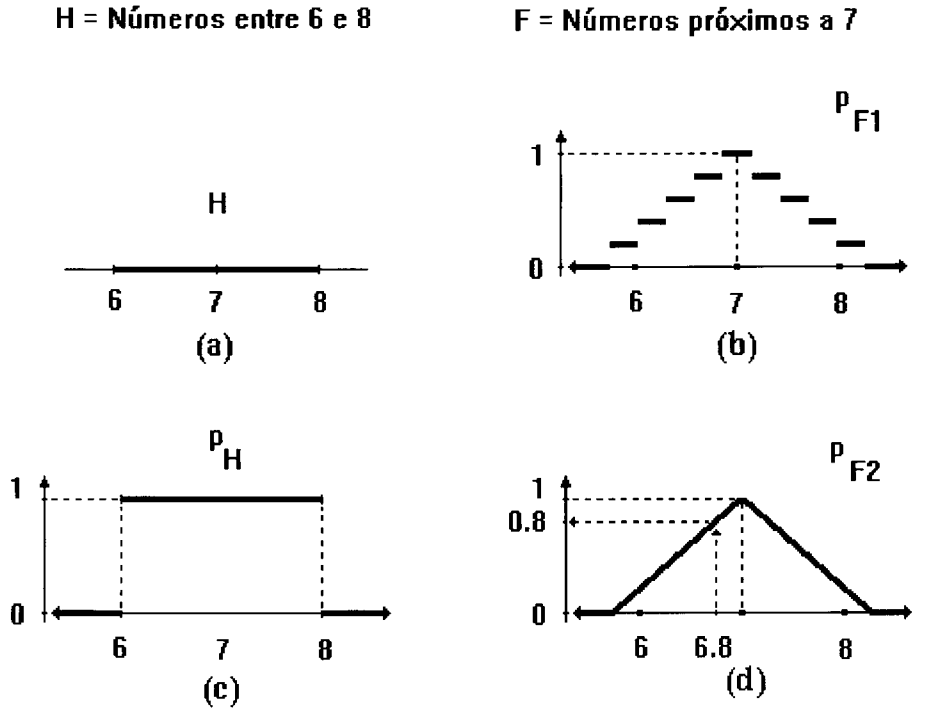


Figura 4.4 - O subconjunto H (a) e as funções de pertinência para os subconjuntos de $\Re$, convencional (c) e fuzzy (b), (d) [BEZD93].

Toda função $p : X \rightarrow [0,1]$ é um conjunto fuzzy (Figura 4.5). Apesar disto ser verdade do ponto de vista da matemática formal, muitas funções que se encaixam nesta definição não podem ser interpretadas convenientemente como uma realização conceitual de um conjunto fuzzy. Em outras palavras, funções que mapeiam X no intervalo unitário *podem* ser conjuntos fuzzy, mas *tornam-se* conjuntos fuzzy quando, e somente quando, elas combinam alguma

descrição semântica plausível de propriedades imprecisas dos objetos em *X* [BEZD93].

Exemplificando, vamos considerar uma classificação de programas por tamanho, medido em linhas de código fonte (LCF). A Figura 4.6 apresenta um gráfico com quatro conjuntos fuzzy representados pelas quatro curvas, da esquerda para a direita, como "Mínimo", "Pequeno", "Médio" e "Grande", respectivamente. Portanto, a variável linguística **Tamanho** pode assumir quatro valores distintos: "**Mínimo**", "**Pequeno**", "**Médio**" e "**Grande**", que são aqui chamados de termos linguísticos. Temos, então, *termo linguístico*: linhas → [0,1]. O mapeamento deste conceito pode ser, por exemplo, o caso no qual um programa com 30 linhas tem um grau de pertinência de 1.0 em **Pequeno**, enquanto que um componente de 55 linhas tem um grau de pertinência de somente 0.7 na mesma classe.

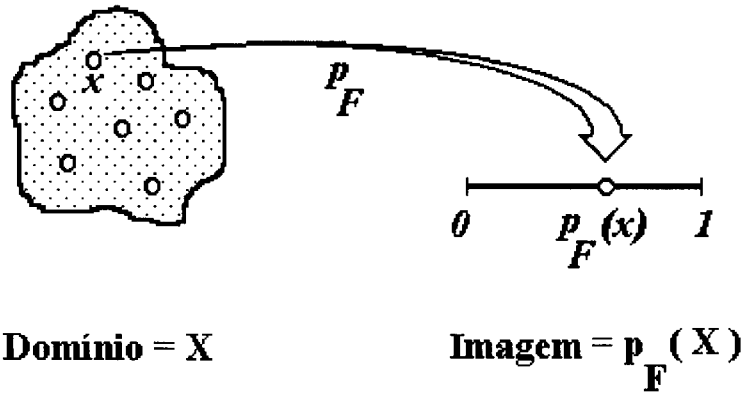


Figura 4.5 - Conjuntos fuzzy são funções de pertinência [BEZD93].

A lógica fuzzy viola o "princípio da não-contradição" no sentido de que, por exemplo, um programa pode ser pequeno e médio (pertencer a dois conjuntos) ao mesmo tempo. É o caso do programa de 55 linhas que tem um grau de pertinência de 0.7 em **Pequeno** e, aproximadamente 0.2 em **Médio**. Temos, portanto, classificações diferentes para um programa ao mesmo tempo.

A escolha de um componente pode ser feita através do grau máximo, ou mínimo, de pertinência de um componente em uma variável linguística. Por exemplo, caso se desejasse o maior grau de pertinência do componente de 55 LCF, do exemplo anterior na variável linguística Tamanho, ele seria classificado como Pequeno, uma vez que o seu grau de pertinência no termo linguístico Pequeno é 0.7. Caso contrário, ou seja, o menor grau, ele seria classificado como Médio, pois o seu grau de pertinência no termo linguístico Médio é aproximadamente 0.2.

Outro critério de classificação pode ser classificar um componente segundo todos os seus graus de pertinência diferentes de zero, em uma variável linguística. Assim o componente de 55 LCF seria classificado como (Pequeno, 0.7) e (Médio, 0.2).

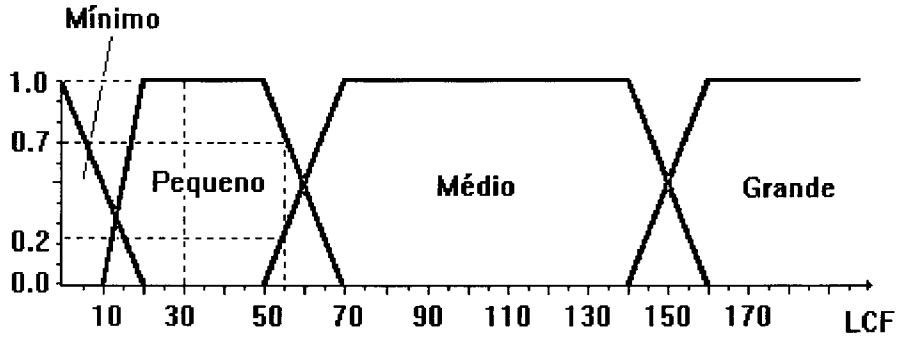


Figura 4.6 - Exemplo de uma variável linguística, **Tamanho**, para classificação do tamanho de um componente em número de linhas de código fonte (LCF). A variável **Tamanho** é definida por quatro termos linguísticos: "**Mínimo**", "**Pequeno**", "**Médio**" e "**Grande**".

Um critério de seleção poderia ser selecionar componentes segundo um grau de pertinência diferente de zero em um termo linguístico específico. Por exemplo, vamos considerar que o universo de componentes é composto pelos componentes de 30 e 55 LCF dos exemplos anteriores. Se fossemos escolher componentes com grau de pertinência diferente de zero para o termo linguístico Pequeno, teríamos como resultado os dois componentes do

universo (30 e 55 LCF). Caso estivéssemos selecionando componentes classificados segundo o termo linguístico Médio, teríamos como resposta somente o componente de 55 LCF.

As variáveis e termos linguísticos são definidos, em número e forma, segundo os resultados que se deseja alcançar. Isto permite uma grande flexibilidade, pois pode-se assim ajustá-los para um aproveitamento máximo em uma determinada situação.

O conceito de lógica fuzzy se insere no contexto deste trabalho no sentido de apresentar um modelo de classificação linguística para componentes de código, em um determinado ambiente de desenvolvimento de software, para reutilização. Esta abordagem permite a criação de um vocabulário particular para identificação e (re)utilização de componentes. Uma vez definidos os termos deste vocabulário, eles podem ser usados como padrão no contexto mais amplo do ambiente, ou seja, nas conversas e reuniões entre gerentes/engenheiros de software/analistas/programadores, nas especificações, na documentação dos sistemas e até na própria codificação dos programas. Esta abordagem permite criar modelos mais próximos à maneira gradual de pensamento das pessoas em relação aos problemas do dia a dia. Segundo esta abordagem um componente, do ponto de vista do gerente, pode ser *um pouco*, *moderadamente* ou *altamente* complexo, por exemplo.

Assim, os valores numéricos podem ser interpretados através de variáveis e termos linguísticos que oferecem uma interpretação mais lógica desses valores numéricos, ainda que relativamente subjetiva. Deve-se levar em consideração o fato de que a interpretação das variáveis e termos linguísticos pode ter significados diferentes para cada usuário, devendo portanto ser seguido um padrão estabelecido para o ambiente ou, pelo menos, diretivas básicas para a definição das variáveis e termos linguísticos.

4.4 A Ferramenta ReFOR

4.4.1 Propósito da Ferramenta

O objetivo da ferramenta é identificar componentes de código FORTRAN candidatos à reutilização a partir da aplicação de métricas de código, seguido de um processo automático de seleção que utiliza variáveis e termos linguísticos, para povoar, em última instância, bibliotecas de componentes reutilizáveis.

A ferramenta utiliza vários tipos de métricas de código, permitindo assim maior flexibilidade de uso e melhores condições de adaptação a diferentes usuários e ambientes de desenvolvimento de software. Apesar de tratar somente da linguagem FORTRAN, considerou-se a necessidade de flexibilidade da ferramenta em função das várias linguagens de programação existentes, pois determinadas métricas são mais apropriadas para determinadas linguagens de programação.

A utilização de variáveis e termos linguísticos tem por objetivo permitir o estabelecimento de uma linguagem padronizada para os valores dos atributos avaliados através da aplicação das métricas de código. Está aproximação de valores numéricos a valores relativamente subjetivos utilizados na linguagem cotidiana, tem por objetivo facilitar a comunicação entre os profissionais da área de desenvolvimento de software através de todas as fases do ciclo de vida do software.

4.4.2 Funções da Ferramenta

A ferramenta está dividida em três módulos: 1) medição, 2) definição das variáveis e termos linguísticos, e, 3) seleção de candidatos à reutilização.

No primeiro módulo os componentes de código são submetidos à medição através de todas as métricas de código implementadas na ferramenta. A

ferramenta permite visualizar os resultados da aplicação das métricas através da sua interface com o usuário e através da emissão de relatórios.

No segundo módulo são criadas as variáveis linguísticas e seus respectivos termos linguísticos. A toda variável linguística é associada uma métrica escolhida pelo usuário. Pode-se aí, também, modificar e excluir variáveis linguísticas e seus respectivos termos linguísticos.

Por fim, no último módulo, são estabelecidos os critérios para seleção de candidatos à reutilização segundo as escolhas feitas pelo usuário. O processo de seleção é ativado pelo usuário. O resultado da seleção pode ser visualizado através de funções da interface gráfica e através de relatórios. A Figura 4.7 apresenta a estrutura da ferramenta ReFOR.

4.4.3 Descrição do Funcionamento

A ferramenta apresenta, na sua janela principal, uma lista com todos os componentes de código que podem ser submetidos à medição. Esses componentes de código fonte são programas, subrotinas e funções, que devem, obrigatoriamente, ter sido previamente compilados. A ferramenta apresenta a relação de todos os componentes que encontrar no diretório alvo. Estes estão classificados em ordem alfabética pelo nome do componente, tipo (FUNCTION, SUBROUTINE ou PROGRAM) e o nome do arquivo DOS que contém o componente (este foi um artifício utilizado devido à incapacidade do DOS de aceitar nomes de arquivos maiores do que oito caracteres alfanuméricos). A medição pode ser efetuada individualmente (por componente) ou todos de uma só vez.

O usuário pode, então, executar a medição de um componente (o selecionado na lista de componentes) ou pode escolher executar a medição de todos os componentes da lista. O sistema sinaliza quando acabou de medir um componente (ou todos) e marca, na lista de componentes, aqueles que foram processados. Pode-se, então, visualizar (ou imprimir) os resultados da avaliação segundo a opção escolhida no menu. Nesta primeira etapa os componentes (ou componente) são submetidos às métricas e critérios

estabelecidos e os resultados são os valores numéricos das medidas. O usuário pode, ainda nesta etapa, solicitar um relatório geral, ou individual, sobre os componentes avaliados. Estes relatórios tem por objetivo apresentar ao usuário um plano geral das medidas efetuadas, fornecendo subsídios para a criação das variáveis linguísticas.

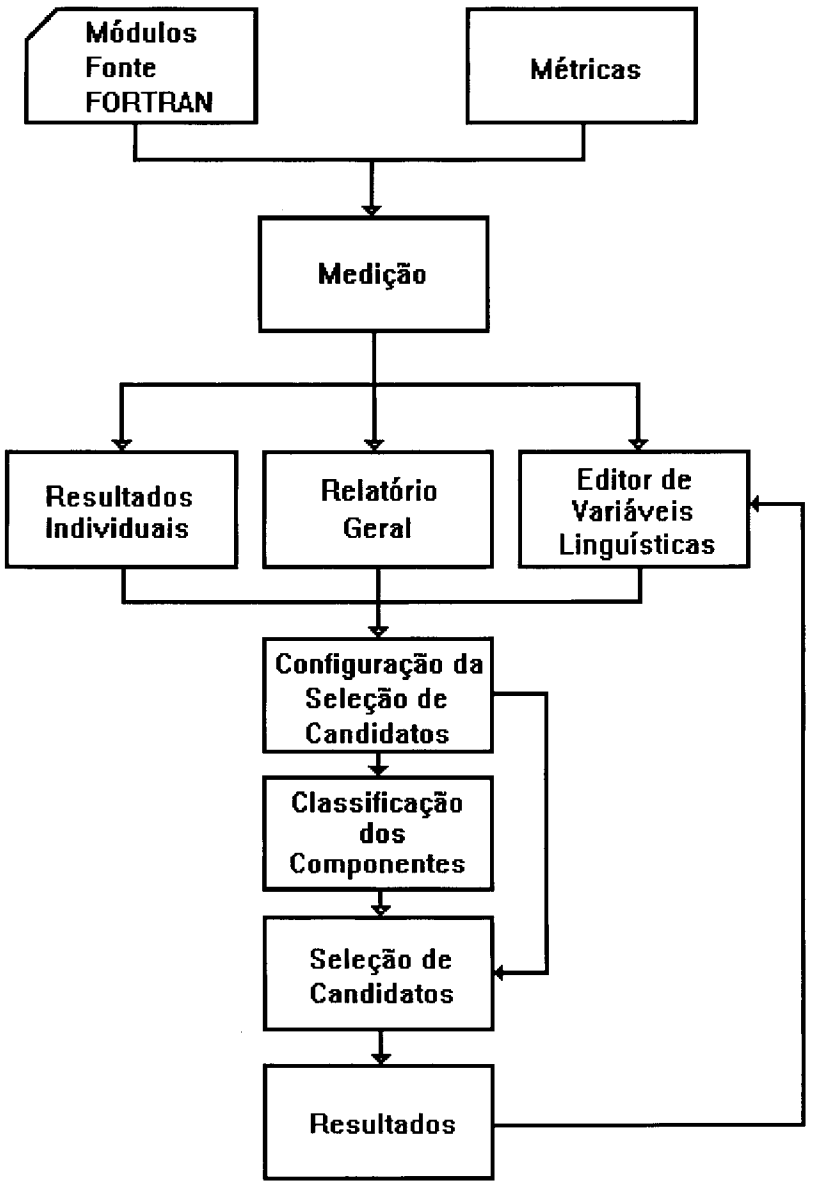


Figura 4.7 - Estrutura da ferramenta ReFOR.

Para criação das variáveis linguísticas, o usuário utiliza o editor de variáveis linguísticas através de opção do menu. Para criar uma variável linguística nova, o usuário deve fornecer o nome da variável, a escala do eixo das abcissas (o intervalo de variação desejado), a unidade de medida da variável e associar uma métrica à variável, dentre as disponíveis na lista apresentada. O eixo das ordenadas (y) tem escala fixa de zero a um (0,1), pois é nele que será identificado o grau de pertinência. A partir deste ponto podem ser criados os termos linguísticos da variável linguística. Para criar um termo linguístico é necessário fornecer o nome do termo linguístico e os valores numéricos da abcissa em função dos vértices da curva escolhida. Cada termo linguístico criado é apresentado em uma cor diferente e, é inserido na lista de termos linguísticos.

Após criar as variáveis linguísticas, o usuário pode configurar a seleção de candidatos à reutilização segundo suas necessidades. Ele pode, então, selecionar as variáveis linguísticas desejadas e o(s) respectivo(s) termo(s) linguístico(s) escolhido(s). Pode-se, aqui, estabelecer qual o tipo de operação que deve ser utilizada para a seleção. Estas operações podem ser *mínimo*, *máximo*, *normal* ou *todos*. A operação *mínimo* estabelece que se um componente tiver duas classificações, em relação à variável linguística ao mesmo tempo, deve ser escolhida a de menor grau de pertinência no conjunto. O mesmo acontece para o caso da operação *máximo*, só que será selecionado o maior grau de pertinência no conjunto. No caso da operação *normal,* só interessa o termo, ou termos configurados que apresentem um grau de pertinência maior do que zero, não sendo consideradas outras possíveis classificações. A operação *todos* classifica os componentes segundo todos os graus de pertinência do componente, em uma variável linguística, para os quais o grau de pertinência é diferente de zero.

O usuário pode pedir uma classificação de todos os componentes, conforme uma configuração efetuada, para ter uma visão geral dos componentes do ponto de vista das variáveis linguísticas.

Nesta última etapa, pode-se avaliar os resultados através de relatórios gerais. A análise destes relatórios permite efetuar ajustes nas variáveis linguísticas, de forma a otimizar o processo segundo a intenção do usuário, realimentando o processo segundo as suas necessidades.

4.4.4 Métricas Utilizadas pela Ferramenta

A partir dos subfatores e critérios identificados na seção 4.2, selecionamos alguns para implementação no primeiro protótipo da ferramenta ReFOR. São avaliados os seguintes critérios:

a) **Tamanho**

É avaliado através das seguintes métricas:

- Métrica de tamanho (*N*) de Halstead: $N = N_1 + N_2$
- Métrica de volume (*V*) de Halstead: $V = N \times \log_2 n$
- Número de linhas de código: contagem do número de linhas de código, excluídas as linhas em branco e as linhas de comentário.

b) **Regularidade**

É avaliada através da seguinte métrica:

- Métricas de tamanho estimado (*N^*) e de tamanho real (*N*) de Halstead: $r = N^{\wedge} / N$

c) **Complexidade**

É avaliada através das seguintes métricas:

- Número ciclomático de McCabe: número de IF's
- Número ciclomático de McCabe modificada: número de IF's + AND's + OR's + XOR's + NOT's + EQV's + NEQV's

d) **Padronização**

É avaliada através da seguinte métrica:

- Bloco de comentários inicial do componente: o bloco de comentários inicial do componente é avaliado segundo o padrão exigido (nome do autor, data de criação, data e motivo das modificações, descrição do objetivo, descrição dos parâmetros de E/S e descrição das variáveis usadas).

e) **Organização Visual**

É avaliada através da seguinte métrica:

- Escore de zero a dez onde é exigido que:

  1) se tenha uma linha em branco antes e depois de uma chamada (call) de rotina;

  2) uma linha em branco antes e depois de um bloco de comentários;

  3) a numeração dos "labels" deve estar em ordem crescente;

  4) o deslocamento de pelo menos um branco, para esquerda (indentação), nos comandos seguintes a um IF, a um DO e nas linhas de continuação.

  O escore é calculado através da média aritmética dos quatro itens relacionados. Cada um dos quatro itens é calculado através da percentagem de atendimento à exigência e convertido para um valor na escala de zero a dez.

f) **Documentação Interna**

É avaliada através da seguinte métrica:

- Percentagem de linhas de comentário do componente.

g) **Programação Estruturada**

É avaliada através da seguinte métrica:

- Escore de zero a dez onde é exigido:

  1) nenhuma ocorrência do comando ENTRY (cada ocorrência subtrai três do escore inicial dez);

  2) somente um comando RETURN (no caso de subrotinas; cada ocorrência adicional subtrai um do escore inicial);

  3) número mínimo de comandos GOTO (cada ocorrência subtrai dois do escore inicial).

  A partir do escore inicial dez são aplicadas as subtrações relacionadas acima.

h) **Fan-out**

É avaliado através da seguinte métrica:

- Número de componentes chamados: contagem do número de componentes chamados.

i) **Fan-in**

É avaliado através da seguinte métrica:

- Número de vezes que é chamado: contagem do número de vezes que o componente é chamado (a partir do fan-out).

j) **Não Memorização**

É avaliado através da seguinte métrica:

- Escore de zero a dez onde:

  1) cada ocorrência de comandos COMMON (local e bloco) subtrai três do valor inicial dez;

  2) cada ocorrência do comando EQUIVALENCE subtrai dois do valor inicial dez.

4.5 Implementação da Ferramenta

A ferramenta foi desenvolvida na linguagem orientada a objetos ACTOR 3.1 [ACTO91], sob o ambiente operacional WINDOWS/DOS em microcomputador PC 386. Foram criadas e implementadas vinte e três classes de objetos (mais os métodos necessários), a partir das classes existentes no ambiente de programação da linguagem ACTOR 3.1.

A interface homem-máquina da ferramenta foi desenvolvida no ambiente WINDOWS 3.1 [WIND92], de forma a garantir aos usuários uma interface homem-máquina intuitiva e amigável, segundo o padrão SAA-CUA [IBM89].

A ferramenta foi desenvolvida para o tratamento de programas FORTRAN, conforme a sintaxe definida pela Digital Equipment Corporation (DEC) [VAXF84a], [VAXF84b] para o produto VAX FORTRAN 4.0. Este software foi desenvolvido pela DEC de acordo com as normas da American National Standard Programming Language FORTRAN, ANSI X3.9 - 1978 (FORTRAN 77).

Nas caixas de diálogo, sempre que pertinente, são apresentados três botões com funções padronizadas: **Ok**, para confirmar a opção, ou opções, selecionadas pelo usuário no diálogo e retornar o controle para a janela que lhe deu origem; **Cancelar**, para desconsiderar as seleções efetuadas, fechar a

caixa de diálogo e devolver o controle à janela de origem; ?, para fornecer auxílio, através do sistema padrão de ajuda (Help) do ambiente WINDOWS 3.1, na forma de informações explicativas dentro do contexto em questão.

Caso existam componentes no diretório alvo, a ferramenta varre todos os arquivos e apresenta-os na lista da sua janela principal (Figura 4.9). No caso de não haver arquivos no diretório, a lista é apresentada em branco. A Figura 4.8 apresenta a lista de componentes, da janela principal, em branco com a caixa de informação sobre a ferramenta ReFOR.

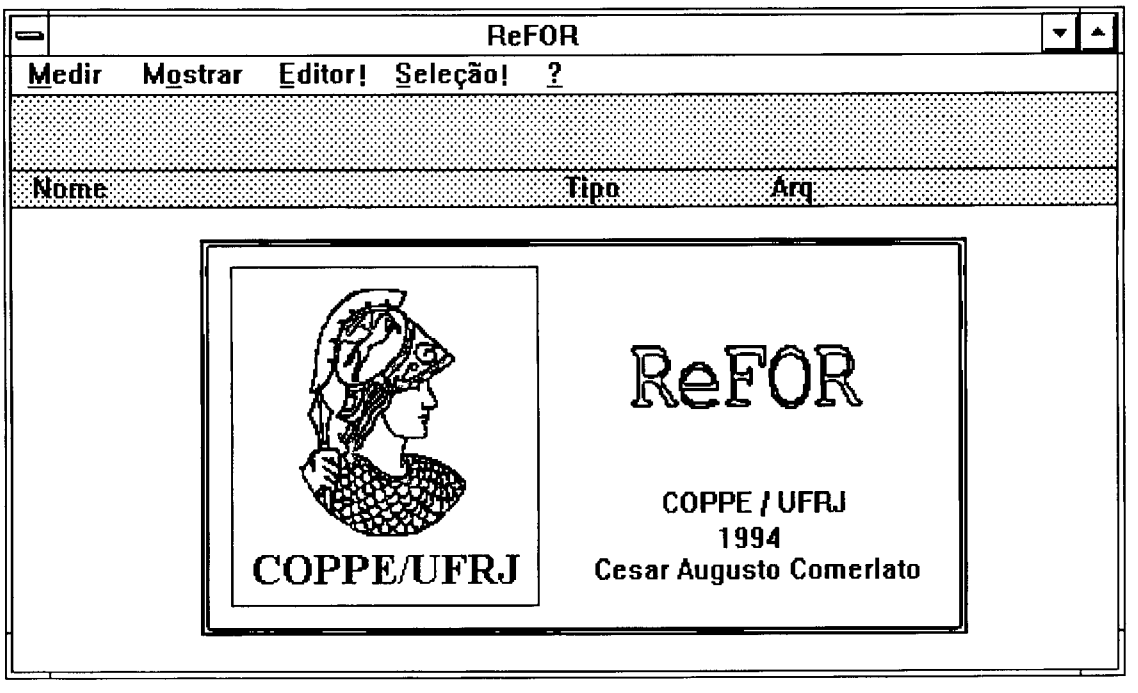


Figura 4.8 - Janela principal da ReFOR na abertura, com a caixa de informação sobre o aplicativo.

Para iniciar a medição, o usuário deve selecionar a opção **Medir** no menu da janela principal. Este item de menu apresenta quatro opções **Componente**, **Todos**, **Executar Relatório Geral** e **Executar Relatório Individual**. A primeira inicia a medição do componente selecionado na lista, a segunda inicia a medição de todos os componentes que ainda não foram medidos, a terceira executa o relatório geral dos componentes que já foram medidos e a quarta executa o relatório individual do componente selecionado na lista. Os

resultados são armazenados internamente (em arquivos) e para visualizá-los deve ser selecionada a opção de menu **Mostrar Resultados das Métricas** e escolher o resultado que se deseja visualizar. Esta operação pode ser acompanhada através da Figura 4.10.

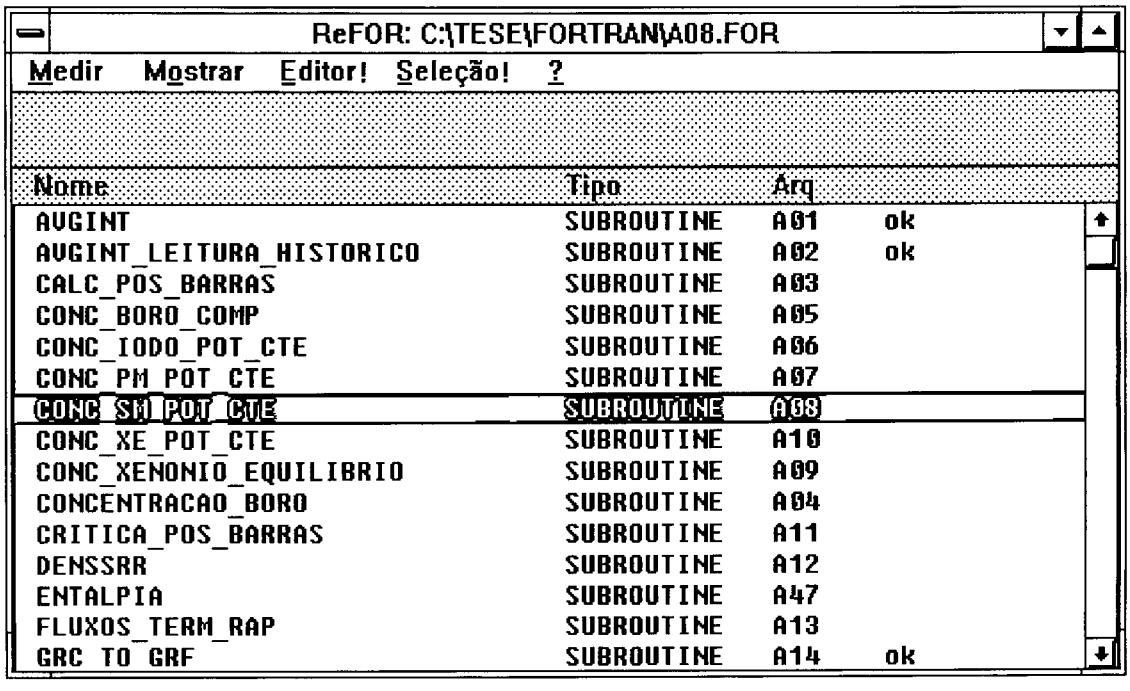


Figura 4.9 - Janela principal da ferramenta ReFOR.

O resultado da opção escolhida (**V, N, N^, N1, N2, n, n1, n2, r**) pode ser visualizado na Figura 4.11. O botão **Sair**, como o próprio nome indica, serve para fechar a janela e devolver o controle para a janela principal, e o botão **Imprimir** imprime um relatório com os dados apresentados.

O item de menu **Mostrar**, da janela principal, apresenta outras duas opções além da opção **Resultados das Métricas**: **Componente** e **Número de Componentes na Lista**. A opção **Componente** , quando selecionada, abre uma janela e lista o código do componente selecionado na lista da janela principal. A opção **Número de Componentes na Lista** apresenta o número de componentes que estão listados.

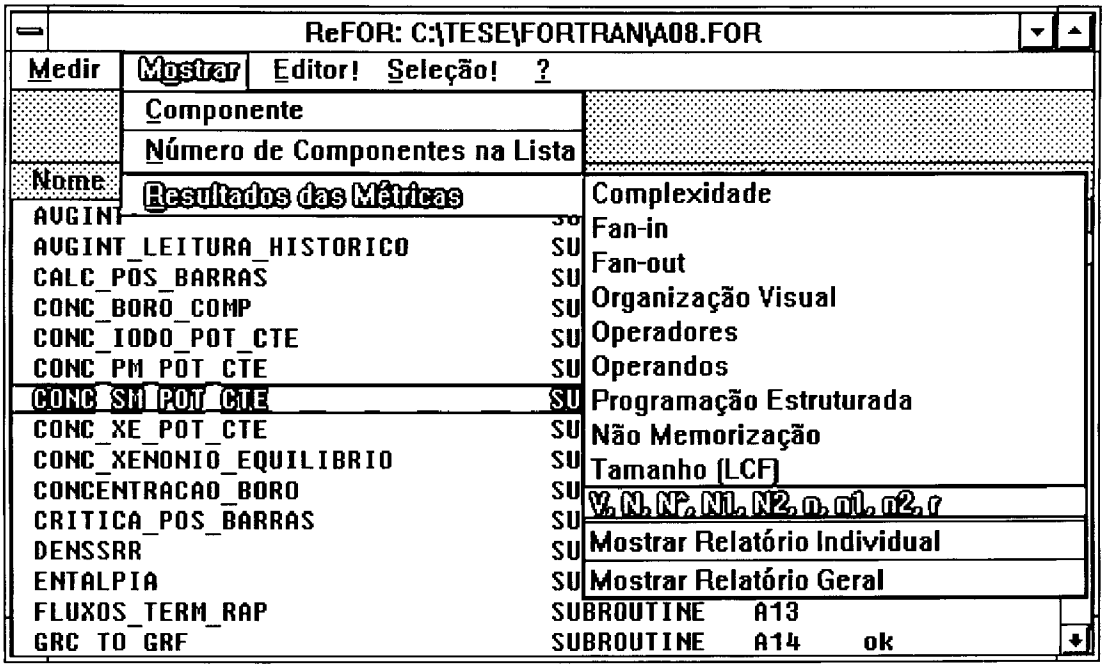


Figura 4.10 - Seleção do item de menu **Mostrar Resultados das Métricas**.

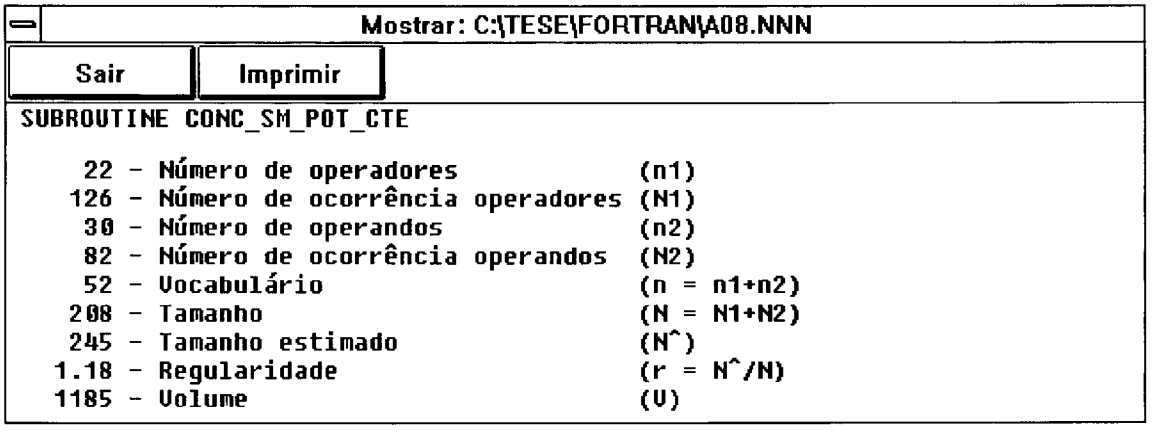


Figura 4.11 - Visualização dos resultados da opção **Mostrar Resultados das Métricas V, N, N^, N1, N2, n, n1, n2, r**.

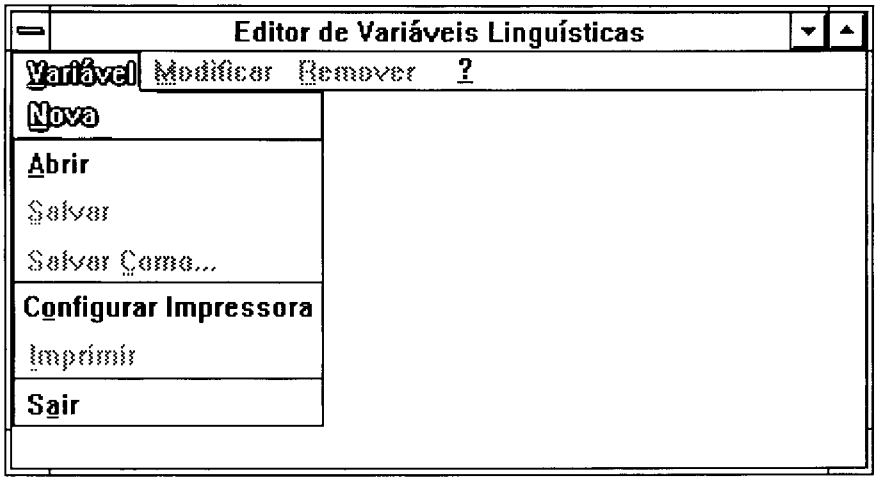


Figura 4.12 - Janela do Editor de Variáveis Linguísticas com o item de menu **Variável** expandido.

As opções de menu **Editor!** e **Seleção!** da janela principal ativam, respectivamente, o Editor de Variáveis Linguísticas e o Módulo de Configuração e Seleção descritos a seguir.

A Figura 4.13 apresenta o diálogo de criação de uma variável linguística nova. O usuário deve informar o nome da variável linguística, a unidade da variável, a métrica associada e a extensão do eixo X. A partir deste ponto o usuário pode usar os quatro botões, dispostos no alto da janela do Editor (Figura 4.14), para criar os termos linguísticos da variável linguística.

A Figura 4.15 apresenta o diálogo de criação de um termo linguístico através da utilização da curva definida pelo segundo botão (da esquerda para a direita). O usuário informa o termo linguístico desejado e informa somente os valores de x das coordenadas dos vértices 1 e 2 da curva, pois os valores de y são fixos, como mostra o ícone desenhado na caixa de diálogo.

Voltando à janela principal, o usuário pode alterar um termo linguístico através da opção de menu **Modificar**. Para tanto, basta selecionar o termo linguístico

que deseja modificar na lista de termos linguísticos e selecionar o item de menu **Modificar**. O usuário pode ainda remover um termo linguístico ou uma variável linguística, e todos seus termos linguísticos, através do item de menu **Remover**. Quando é selecionada a opção **Remover,** abre-se uma caixa de diálogo perguntando se o usuário deseja remover um termo linguístico ou a variável linguística e todos os seus termos linguísticos.

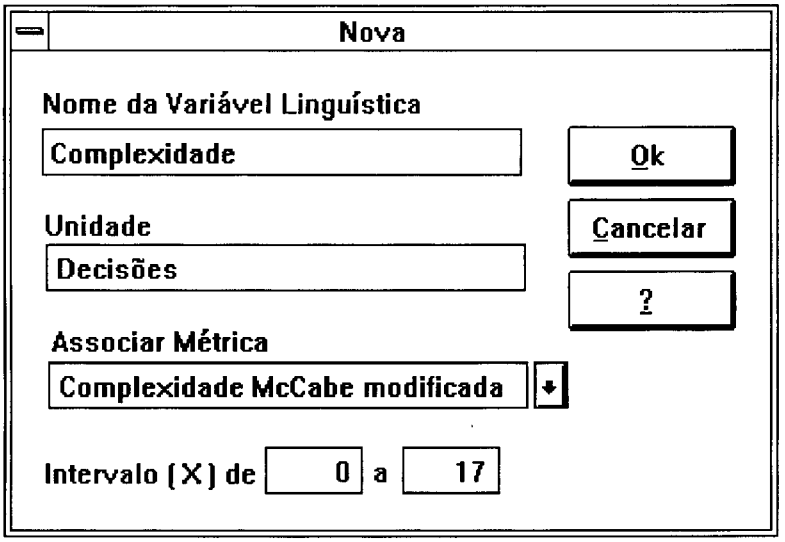


Figura 4.13 - Diálogo de definição de uma **Variável** linguística **Nova**.

A Figura 4.16 apresenta a janela do Módulo de Configuração e Seleção. Nela o usuário vai configurar os critérios de seleção de candidatos à reutilização. Na lista de variáveis linguísticas, o usuário seleciona uma variável e a seguir seleciona o termo, ou termos, linguísticos na lista de termos linguísticos, segundo os quais serão selecionados os candidatos à reutilização, em relação a esta variável linguística. A seguir deve "apertar" o botão **Inserir** para incluir as escolhas feitas na lista de Critérios de Seleção. Caso o usuário queira retirar algum dos critérios incluídos, basta selecioná-lo na lista de Critérios de Seleção e utilizar, a seguir, o botão **Remover**.

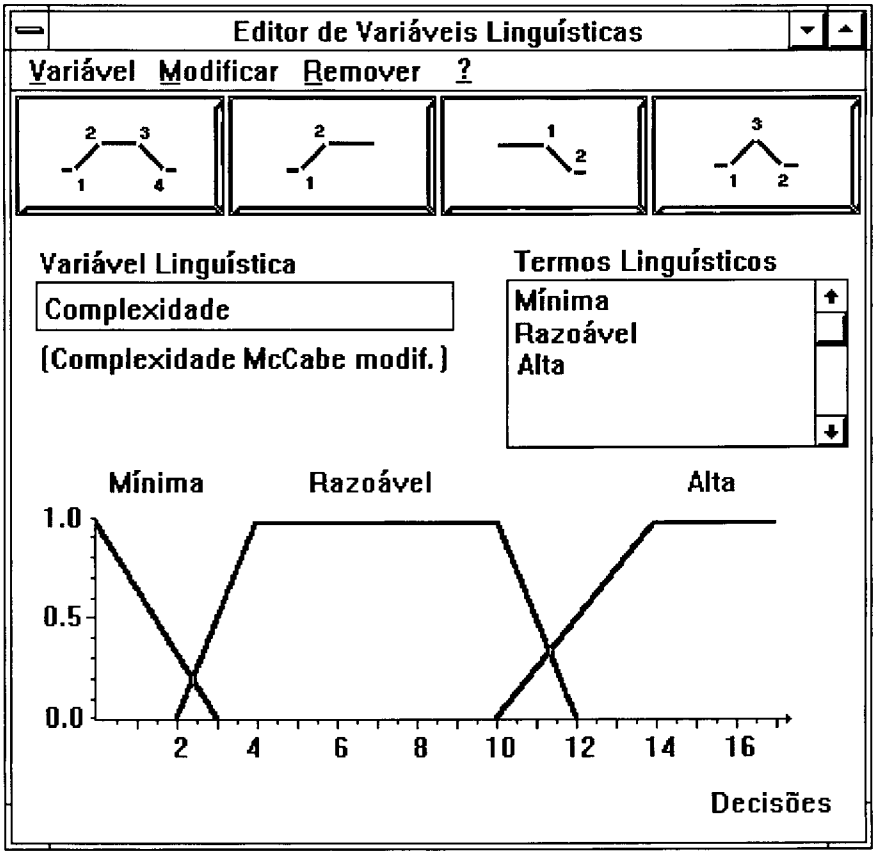


Figura 4.14 - Janela do Editor de Variáveis Linguísticas durante o processo de criação de uma variável linguística.

Uma vez definidos todos os critérios de seleção desejados, o usuário pode utilizar o botão **Selecionar...** para iniciar o processo de seleção. O botão **Classificar...** permite que seja feita somente uma classificação de todos os componentes, segundo os critérios de seleção escolhidos. Nestas duas operações são gerados relatórios que podem ser visualizados ou impressos através da opção de menu **Mostrar**. Esta opção permite também que o usuário visualize as curvas das variáveis linguísticas através da chamada do Editor de Variáveis Linguísticas.

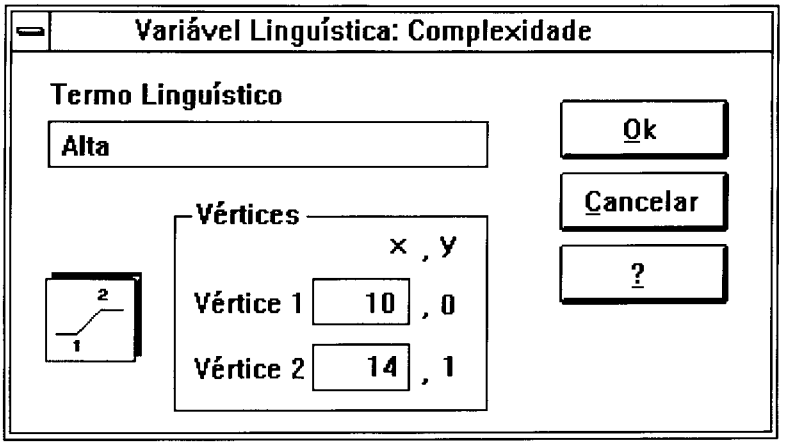


Figura 4.15 - Diálogo de definição de um termo linguístico (Alta) da variável linguística Complexidade utilizando o segundo botão (da esquerda para a direita).

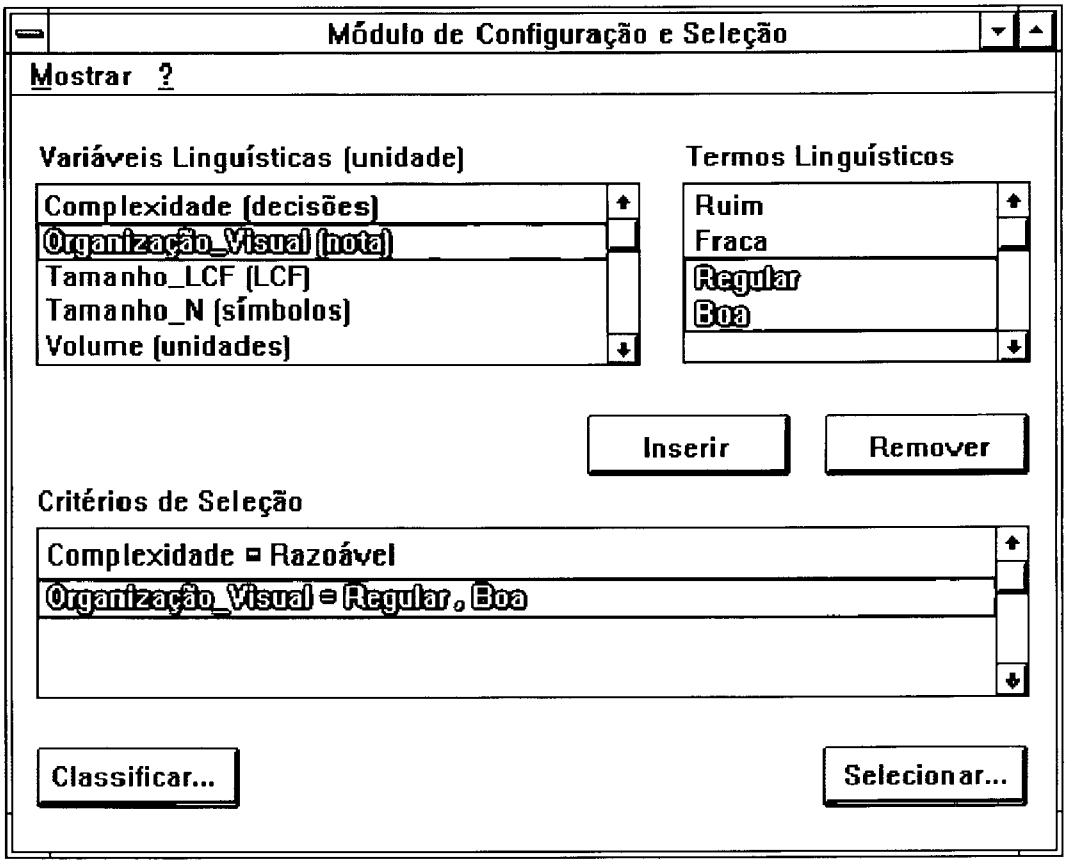


Figura 4.16 - Janela do Módulo de Configuração e Seleção.

4.6 Exemplos de Utilização

O protótipo da ferramenta foi construído para avaliar e selecionar componentes desenvolvidos segundo os padrões, utilizados na época, do ambiente de programação do Laboratório de Monitoração de Processos do Programa de Engenharia Nuclear da COPPE/UFRJ, utilizado para o desenvolvimento e implementação do Sistema Integrado de Computadores de Angra (SICA) no período de 1986 a 1990.

Para validar o uso da ferramenta ReFOR foram avaliados quarenta e nove componentes (subrotinas e programas) de um subsistema do Sistema Integrado de Computadores de Angra (SICA). Este subsistema foi escolhido por apresentar o código FORTRAN mais "puro", isto é, com o mínimo de chamadas de funções do sistema operacional, gerenciadores de tela e banco de dados. O objetivo deste subsistema é executar uma série de cálculos de aplicações nucleares. A Figura 4.17 apresenta um exemplo de uma subrotina FORTRAN.

```
C*=====================================================================*
C*                                                                     *
      SUBROUTINE GRF_TO_GRC (          GR_FARENHEIT            ,
     &                                 STS_GR_FARENHEIT        ,
     &                                 GR_CELSIUS              ,
     &                                 STS_GR_CELSIUS          )
C*                                                                     *
C*=====================================================================*

      IMPLICIT INTEGER*4 ( A - Z )

C*=====================================================================*
C*                                                                     *
C*    LABORATORIO DE MONITORACAO DE PROCESSOS                          *
C*                                                                     *
C*    SICA - SISTEMA INTEGRADO DE COMPUTADORES DE ANGRA I              *
C*                                                                     *
C*=====================================================================*
C*                                                                     *
C*    ESTE PROGRAMA FOI DESENVOLVIDO PELA COPPE/UFRJ - COORDENACAO     *
C*    DOS PROGRAMAS DE POS-GRADUACAO EM ENGENHARIA DA UNIVERSIDADE     *
C*    FEDERAL DO RIO DE JANEIRO NO PROGRAMA DE ENGENHARIA NUCLEAR.     *
C*                                                                     *
C*=====================================================================*
C*                                                                     *
C*    AUTOR : IVO WOLFF GERSBERG                  DATA : 14/06/89      *
C*                                                                     *
C*=====================================================================*
```

Figura 4.17 - Subrotina FORTRAN para conversão de graus Farenheit para graus Celsius.

```
C*    OBJETIVO : Faz a conversao de graus farenheit para celsius.      *
C*                                                                     *
C*=====================================================================*
C*                                                                     *
C*    ALTERACOES                                                       *
C*                                                                     *
C*     AUTOR                  DATA          MOTIVO                     *
C*     -----                  ----          ------                     *
C*                                                                     *
C*     XXXXXXXXXXXXXX         XX/XX/XX      XXXXXXXXXXXXXXXXXXXXXXXX   *
C*                                          XXXXXXXXXXXXXXXXXXXXXXXX   *
C*                                                                     *
C*=====================================================================*
C*                                                                     *
C*    DESCRICAO DOS ARGUMENTOS DE ENTRADA E SAIDA                      *
C*                                                                     *
C*     NOME                   DESCRICAO                                *
C*     ----                   ---------                                *
C*                                                                     *
C*     GR_CELSIUS             Valor em celsius para a conversao.       *
C*                                                                     *
C*     STS_GR_CELSIUS         Status da variavel em celsius ( entrada ).*
C*                                                                     *
C*     GR_FARENHEIT           Valor convertido a retornar.             *
C*                                                                     *
C*     STS_GR_FARENHEIT       Status do valor convertido a  retornar.  *
C*                                                                     *
C*=====================================================================*
C*                                                                     *
C*    INCLUDE DOS PARAMETROS :                                         *
C*                                                                     *
       INCLUDE 'P2500$PARAMETROS:SISTEMA.INC'
C*                                                                     *
C*                     FIM DOS BLOCOS DE PARAMETROS                    *
C*                                                                     *
C*=====================================================================*

       BYTE            STS_GR_CELSIUS                  ,
     &                 STS_GR_FARENHEIT

       REAL*4          GR_CELSIUS                      ,
     &                 GR_FARENHEIT

C      ------------------------------------------
C      verifica se variavel em celsius esta' valida.
C      ------------------------------------------

       IF( STS_GR_FARENHEIT .EQ. INVALIDO ) THEN

           GR_CELSIUS       = 0.0
           STS_GR_CELSIUS   = INVALIDO

       ELSE

C          ------------------
C          calcula conversao.
C          ------------------

           GR_CELSIUS     = ( ( GR_FARENHEIT - 32. ) * 5. ) / 9.
           STS_GR_CELSIUS = VALIDO

       END IF

       RETURN
       END
```

Figura 4.17 (cont.) - Subrotina para conversão de graus Farenheit para graus Celsius.

A Figura 4.18 apresenta o relatório individual das medidas feito na primeira etapa da ferramenta ReFOR.

```
Relatório Individual: SUBROUTINE GRF_TO_GRC
Linhas de Código

          19  linhas de código fonte
           4  linhas de comentário não-alfanumérico
           2  linhas de comentário com palavras
          60  linhas de comentário no bloco inicial
          66  linhas de comentário (total)
          19  linhas em branco
         104  linhas (total do componente)

        18.3 % de linhas em branco
        63.5 % de linhas de comentário
        18.3 % de linhas de código fonte

Complexidade
     2 - Número ciclomático de McCabe
     2 - Número ciclomático de McCabe modificado

Métricas de Halstead
     20 - Número de operadores            (n1)
     45 - Número de ocorrência operadores (N1)
     14 - Número de operandos             (n2)
     25 - Número de ocorrência operandos  (N2)
     34 - Vocabulário                     (n = n1+n2)
     70 - Tamanho                         (N = N1+N2)
    139 - Tamanho estimado                (N^)
     2. - Regularidade                    (r = N^/N)
    356 - Volume                          (V)

Operadores e ocorrências
*                   1
&                   5
,                   5
/                   1
-                   2
=                   4
.EQ.                1
BYTE                1
ELSE                1
END IF              1
END                 1
IF                  1
IMPLICIT            1
INCLUDE             1
INTEGER*4           1
REAL*4              1
RETURN              1
SUBROUTINE          1
THEN                1
fim-de-linha       14
--------------------
n1= 20         N1= 45
```

Figura 4.18 - Relatório individual das medidas efetuadas pela ferramenta ReFOR da subrotina GRF_TO_GRC (Figura 4.17).

```
Operandos e ocorrências
A                                       1
Z                                       1
'P2500$PARAMETROS:SISTEMA.INC'          1
9.                                      1
32.                                     1
VALIDO                                  1
5.                                      1
GRF_TO_GRC                              1
0.0                                     1
INVALIDO                                2
GR_FARENHEIT                            3
STS_GR_FARENHEIT                        3
GR_CELSIUS                              4
STS_GR_CELSIUS                          4
-----------------------------------------
n2= 14                              N2= 25

Fan-out
Não chama nenhuma rotina

Organização Visual
8.6 - Organização Visual (0 a 10)

Não-Memorização
10. - Não-Memorização (0 a 10)

Programação Estruturada
10. - Programação Estruturada (0 a 10)
```

Figura 4.18 (cont.) - Relatório individual das medidas efetuadas pela ferramenta ReFOR da subrotina GRF_TO_GRC (Figura 4.17).

A Figura 4.23 apresenta uma classificação de dez componentes segundo as variáveis linguísticas apresentadas nas Figuras 4.19, 4.20, 4.21 e 4.22.

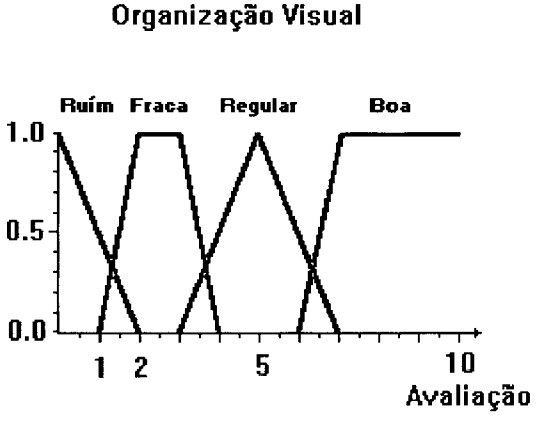


Figura 4.19 - Variável e termos linguísticos para a métrica de avaliação da organização visual.

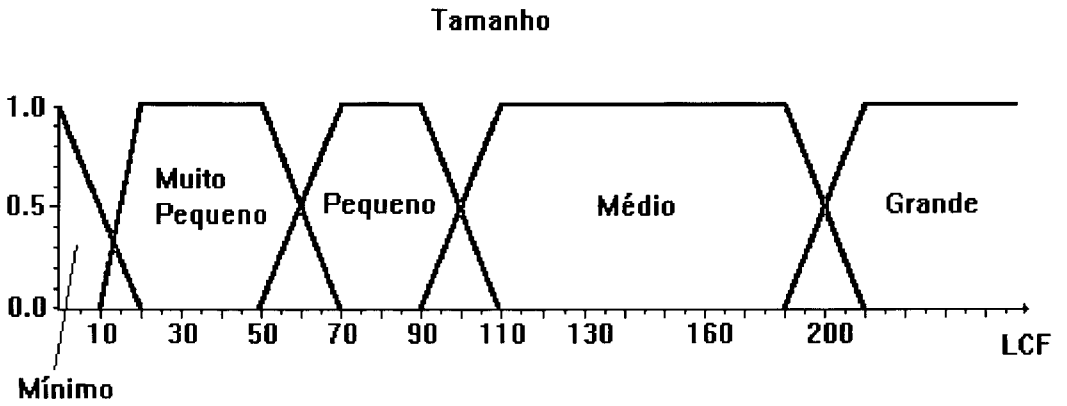


Figura 4.20 - Variável e termos linguísticos definidos para a métrica de número de linhas de código fonte executável.

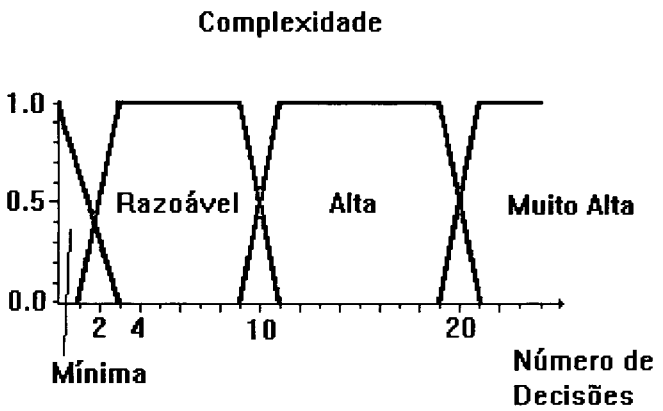


Figura 4.21 - Variável e termos linguísticos definidos para a métrica de complexidade modificada de McCabe.

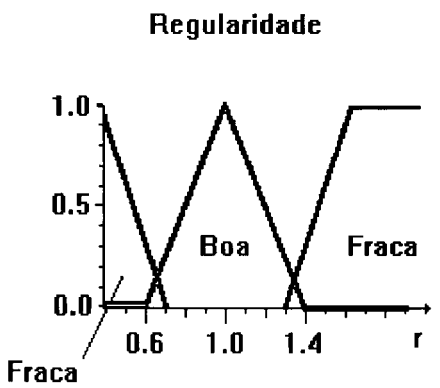


Figura 4.22 - Variável e termos linguísticos definidos para a métrica de regularidade.

```
Relatório de Classificação
Número de componentes classificados: 10

Componente SUBROUTINE AVGINT
Organização visual = Boa (9.2)
Tamanho (LCF) = Médio (157 LCF)
Complexidade = Muito Alta (26)
Regularidade = Boa (0.98)

Componente  SUBROUTINE AVGINT_LEITURA_HISTÓRICO
Organização visual = Boa (8.8)
Tamanho (LCF) = Grande (264 LCF)
Complexidade = Muito Alta (55)
Regularidade = Boa (0.80)

Componente SUBROUTINE CALC_POS_BARRAS
Organização visual = Boa (8.9)
Tamanho (LCF) = Pequeno (74 LCF)
Complexidade = Razoável (9)
Regularidade = Boa (0.74)

Componente SUBROUTINE CONC_BORO_COMP
Organização visual = Boa (8.6)
Tamanho (LCF) = Muito_Pequeno (34 LCF)
Complexidade = Razoável (3)
Regularidade = Fraca (1.49)

Componente SUBROUTINE CONC_IODO_POT_CTE
Organização visual = Boa (7.3)
Tamanho (LCF) = Muito_Pequeno(35 LCF)
Complexidade = Razoável (4)
Regularidade = Fraca (1.51)
```

Figura 4.23 - Resultados da classificação dos componentes segundo as variáveis e termos linguísticos das Figuras 4.19, 4.20, 4.21 e 4.22.

```
Componente SUBROUTINE CONC_PM_POT_CTE
Organização visual = Boa (7.3)
Tamanho (LCF) = Muito_Pequeno(35 LCF)
Complexidade = Razoável (4)
Regularidade = Fraca (1.51)

Componente SUBROUTINE CONC_SM_POT_CTE
Organização visual = Boa (7.9)
Tamanho (LCF) = Muito_Pequeno(52 LCF)
Complexidade = Razoável (5)
Regularidade = Boa (1.18)

Componente SUBROUTINE CONC_XE_POT_CTE
Organização visual = Boa (7.6)
Tamanho (LCF) = Muito_Pequeno(56 LCF)
Complexidade = Razoável (5)
Regularidade = Boa (1.19)

Componente SUBROUTINE CONC_XENONIO_EQUILIBRIO
Organização visual = Boa (6.9)
Tamanho (LCF) = Muito_Pequeno(23 LCF)
Complexidade = Razoável (2)
Regularidade = Fraca (1.97)

Componente SUBROUTINE CONCENTRACAO_BORO
Organização visual = Boa (8.6)
Tamanho (LCF) = Muito_Pequeno(82 LCF)
Complexidade = Razoável (7)
Regularidade = Boa (1.12)
```

Figura 4.23 (cont.) - Resultados da classificação dos componentes segundo as variáveis e termos linguísticos das Figuras 4.19, 4.20, 4.21 e 4.22.

Se considerarmos que componentes candidatos à reutilização devem atender aos seguintes critérios:

- Organização visual = Boa
- Tamanho (LCF) = Pequeno
- Complexidade = Razoável
- Regularidade = Boa

O resultado entre os dez componetes classificados seria a seleção de um só componente, a subrotina CALC_POS_BARRAS. A Figura 4.24 apresenta o resultado da seleção de candidatos à reutilização segundo os critérios estabelecidos acima.

```
Relatório de Seleção de Candidatos
Número de componentes avaliados: 10
Número de componentes selecionados: 1

Componente selecionados

SUBROUTINE CALC_POS_BARRAS
```

Figura 4.24 - Resultado da seleção de componentes segundo os critérios estabelecidos.

5 CONCLUSÕES

5.1 Considerações Finais

Neste trabalho foi desenvolvida uma ferramenta para identificação de componentes FORTRAN candidatos à reutilização, segundo medidas obtidas através de métricas de software e seleção, através de variáveis linguísticas e termos linguísticos pré-determinados.

Uma das contribuições deste trabalho é a definição de subfatores, critérios e métricas para avaliar o potencial de reutilizabildade de código. Outra contribuição é a utilização, na ferramenta, de variáveis e termos linguísticos para identificação de componentes de código candidatos à reutilização. Foi também definido um processo de avaliação de candidatos à reutilização baseado na sequência de três atividades: medição, definição de variáveis e termos linguísticos, que representam critérios de qualidade, e a seleção de candidatos através destas variáveis.

5.2 Sugestões para Futuros Trabalhos

O objetivo do desenvolvimento da ferramenta ReFOR foi demonstrar a viabilidade prática da proposta desta tese. A ferramenta deve, portanto, passar por outros "casos" testes para ser validada e aperfeiçoada.

A ferramenta pode ser melhorada de várias formas, todas elas visando aumentar sua capacidade, flexibilidade e facilidade de uso, com o objetivo de se obter maiores benefícios para a prática da reutilização. Por exemplo, poderia ser criado um banco de dados de métricas, com uma interface homem-

máquina interativa que permitisse ao usuário configurar a etapa de medição segundo grupos de métricas particulares. A manutenção do banco de dados permitiria a inclusão de novas métricas, aumentando assim a capacidade de medição da ferramenta.

Outra melhoria possível de ser efetuada é a separação do analisador estático da ferramenta. Poderia-se, assim, configurar a ferramenta para avaliar componentes desenvolvidos em outras linguagens, desde que existissem os respectivos analisadores estáticos.

Uma apresentação gráfica da estrutura dos componentes submetidos ao processo de medição e avaliação seria de grande valor, pois permitiria ao usuário visualizar a estrutura de implementação dos componentes dos sistemas a serem processados.

O editor de variáveis linguísticas pode ser aperfeiçoado através da inclusão de outros modelos de curvas, permitindo assim uma maior flexibilidade de uso para as necessidades específicas dos usuários.

Para que a ferramenta alcance seus objetivos, é indispensável que ela seja adaptável às necessidades e objetivos de ambientes particulares. Portanto, outro aperfeiçoamento poderia ser a capacidade de trabalho compartilhado. Desta maneira, usuários diferentes, ou grupos de usuários, poderiam criar configurações particulares dentro de um mesmo ambiente.

A ferramenta pode ter o seu modelo de avaliação da qualidade de programas expandido para outros fatores de qualidade, além do fator reutilizabilidade. Isto permitiria uma avaliação mais abrangente e completa da qualidade dos componentes.

O protótipo da ferramenta ReFOR desenvolvido trata somente da fase de identificação e seleção de candidatos à reutilização. A qualificação, encapsulamento e técnicas de armazenamento e recuperação são propostas para continuação deste trabalho.

REFERÊNCIAS

[ACTO91] WHITEWATER/SYMANTEC; Actor Programming; 1991.

[ANDR91] C.J. ANDRADE; Avaliação da Qualidade de Programas; Tese de M.Sc. em Engenharia de Sistemas e Computação; COPPE/UFRJ, 1991.

[APPL91] D. APPLEBY; FORTRAN - Classic Languages Part I; Revista BYTE, Setembro 1991.

[ARAN89] G. ARANGO; Domain Analysis: From Art Form to Engineering Discipline; Proc. Fifth International Workshop on Software Specification and Design; pp. 152-159; Maio 1989.

[ARTH85] L. J. ARTHUR; Measuring Programmer Productivity and Software Quality; John Wiley & Sons, 1985.

[BAIL92] S. C. BAILIN, S. HENDERSON; Towards A Case-Based Software Engineering Environment; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[BALD90] D.M. BALDA, D.A. GUSTAFSON; Cost Estimation Models for the Reuse and Prototype Software Development Life-Cycles; ACM Software Engineering Notes Vol. 15 n° 3 pp. 42-50; Julho 1990.

[BARN91] B.H. BARNES, T.B. BOLLINGER; Making Reuse Cost-Effective; IEEE Software pp. 13-23; Janeiro 1991.

[BASI83] V.R. BASILI, R. W. SELBY,T. PHILLIPS; Metric Analysis and Data Validation Across Fortran Programs; IEEE Transactions on Software Engineering, Vol. SE-9, N. 6, Novembro 1983.

[BASI90] V.R. BASILI; Viewing Maintenance as Reuse-Oriented Software Development; IEEE Software, pp. 19-25, Janeiro, 1990.

[BASI92] V.R. BASILI, G. CALDIERA, G. CANTONE; A Reference Architecture for the Component Factory; ACM Transactions on Software Engineering and Methodology, Vol. 1 n° 1 pp. 53-80; Janeiro, 1992.

[BELC92] A. D. BELCHIOR; Controle da Qualidade de Software Financeiro; Tese de M.Sc. em Engenharia de Sistemas e Computação; COPPE/UFRJ, 1992.

[BERS91] E.H. BERSOFF, A.M. DAVIS; Impacts of Life Cycle Models on Software; Communications of the ACM Vol. 34 n° 8 pp. 104-117; Agosto 1991.

[BEZD93] J. BEZDEK; Fuzzy Models - What Are They, and Why? (Editorial); IEEE Transactions on Fuzzy Systems, Vol. 1, No. 1; Fevereiro, 1993.

[BIGG87] T.J. BIGGERSTAFF, C. RICHTER; Reusability Framework, Assessment, and Directions; IEEE Software pp. 41-49, Março 1987.

[BIGG89a] T.J. BIGGERSTAFF, A.J. PERLIS; Software Reusability: Vol.I, Concepts and Models; Addison-Wesley Publishing Company; 1989.

[BIGG89b] T.J. BIGGERSTAFF, A.J. PERLIS; Software Reusability: Vol.II, Applications and Experience; Addison-Wesley Publishing Company; 1989.

[BOEH78] B. W. BOEHM et al; Characteristics of Software Quality; Amsterdam, North-Holland;1978.

[BOOC91] G. BOOCH; Object Oriented Design with Applications; Benjamin/Cummings Publishing Company; 1991.

[BURT87] B. A. BURTON et al; The Reusable Software Library; IEEE Software, pp. 25-33; Julho 1987.

[CALD91] G. CALDIERA, V. R. BASILI; Identifying and Qualifying Reusable Software Components; Computer pp. 61-70; Fevereiro 1991.

[CHIV85] I. D. CHIVERS, M. W. CLARK; History and Future of Fortran; Systems, Vol.27 N.1, pp 39-41, Janeiro / Fevereiro 1985.

[CLEA88] J.C. CLEAVELAND; Building Application Generators; IEEE Software, Julho 1988.

[COHE91] S. COHEN; Process and Products for Software Reuse and Domain Analysis; Proc. Fourth Annual Workshop on Software Reuse; Novembro, 1991.

[CONT86] S.D. CONTE, H.E. DUNSMORE e V.Y.SHEN; Software Engineering Metrics and Models; Benjamim/Cummings Pub Company; 1986.

[COX87] B. COX; Object Oriented Programming: An Evolutionary Approach; Addison-Wesley Publishing Company; 1987.

[CROS86] M. CROSS, A. O. MOSCARDINI, B. A. LEWIS; Software Engineering Methologies for Scientific and Engineering Computation; Appl. Math. Modeling, Vol. 10, Outubro 1986.

[DASK92] M. K. DASKALANTONAKIS; A Practical View of Software Measurement and Implementation Experiences whitin Motorola; IEEE Transactions on Software Engineering, Vol. 18, N. 11, Novembro 1992.

[DAVI91] M. J. DAVIS; STARS Framework for Reuse Process; Proc. Fourth Annual Workshop on Software Reuse; Novembro, 1991.

[DAVI92] M. J. DAVIS; STARS Reuse Maturity Model: Guidelines for Reuse Strategy Formulation; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[DUNN91] M.F. DUNN, J.C KNIGTH; Software Reuse in an Industrial Setting: A Case Study, Proceedings of the 13th International Conference on Software Engineering; pp. 329-338; 1991.

[DUNN93] M.F. DUNN, J.C KNIGTH; Automating the Detection of Reusable Parts in Existing Software, Proceedings of the 15th International Conference on Software Engineering; pp. 381-390; 1993.

[FAGE92] J. FAGET, J. MOREL; The REBOOT Approach to the Concept of a Reusable Component; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[FARI91] E.C. FARIA; MARTE: Um Meta-gerador de Aplicações Baseado na Reutilização de Templates; Tese de M.Sc em Engenharia de Sistemas e Computação; COPPE/UFRJ, 1991.

[FERN92] C. FERNSTROM, K. NARFELT, L. OHLSSON; Software Factory Principles, Architecture, and Experiments; IEEE Software pp. 36-44; Março 1992.

[FISC91] G. FISCHER, S. HENNINGER, D. REDMILES; Cognitive Tools for Locating and Comprehending Software Objects for Reuse; Proceedings of the 13th International Conference on Software Engineering; pp. 318-328; 1991.

[FORT92] - ; From the Editor; Fortran Forum - SIGPLAN Special Interest Publication on Fortran; ACM Press; Vol.11 N.2 (S.N. 30), Junho 1992.

[FRAK91] W.B. FRAKES, et al; Software Reuse: Is it Delivering?; Proceedings of the 13th International Conference on Software Engineering; pp. 52-59; 1991.

[GOLD83] A. GOLDBERG, ROBSON, D.; SMALLTALK-80: The Language and Its Implementation; Addison-Wesley, 1983.

[HALS77] M. H. HALSTEAD; Elements of Software Science; Elsevier North-Holland, 1977.

[HISL91] G. W. HISLOP; Estimating the Potential for Reuse; Proc. Fourth Annual Workshop on Software Reuse; Novembro, 1991.

[HORO89] E. HOROWITZ, J.B. MUNSON; An Expensive View of Reusable Software; em Software Reusability: Vol.I, Concepts and Models; T.J. BIGGERSTAFF e A.J. PERLIS (editores), Addison-Wesley Publishing Company; 1989.

[IBM89] IBM Corporation; Systems Application Architecture (SAA) Common User Access (CUA) Advanced Interface Design Guide; Document Number: SY0328-300-R00-1089; Primeira Edição, Junho 1989.

[ISCO91] N. ISCOE, G.B. WILLIANS, G. ARANGO; Domain Modeling for Software Engineering; Proceedings of the 13th International Conference on Software Engineering; pp. 340-343; 1991.

[ISOD92] S. ISODA; Experience Report on Software Reuse Project: Its Structure, Activities, and Statistical Results; Proceedings of the 14th International Conference on Software Engineering; pp. 320-326; 1992.

[JONE91] C. JONES; Applied Software Measurement- Assuring Productivity and Quality; McGraw Hill, Inc.; 1991.

[KARL92] E. KARLSSON; A Cleanroom Approach to Object Oriented Development for Reuse; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[KARR91] C. KARR; Genetic Algorithms for Fuzzy Controlers; AI EXPERT, Fevereiro 1991.

[LANE89] R. G. LANERGAN, C. A. GRASSO; Software Engineering with Reusable Designs and Code; em Software Reusability: Vol.II, Applications and Experience; T. J. BIGGERSTAFF e A. J. PERLIS (editores), Addison-Wesley Publishing Company; 1989.

[LANG93] N. LANG; Shlaer-Mellor Object-Oriented Analysis Rules; ACM Sigsoft Software Engineering Notes, Vol. 18 n° 1 pp. 54-58; Janeiro 1993.

[LEA92] D. LEA, D. CHAMPEAUX; Object Oriented Software Reuse Technical Opportunities; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[LENZ87] M. LENZ, H. A. SCHMID, P. F. WOLF; Software Reuse through Building Blocks; IEEE Software, pp. 34-42; Julho 1987.

[LEWI91] J.A. LEWIS, S.M. HENRY, D.G. KAFURA, R.S. SCHULMAN; An Empirical Study of the Object-Oriented Paradigm and Software Reuse; OOPSLA'91, pp. 184-196, 1991.

[LIAO93] HSIAN-CHOU LIAO, FENG-JIAN WANG; Software Reuse Based on a Large Object-Oriented Library; ACM Software Engineering Notes Vol. 18 n° 1 pp. 74-78; Janeiro 1993.

[LIVS91] B.LIVSON; Software Chips DataBase; ACM Software Engineering Notes Vol. 16 n° 1 pp. 41-42; Janeiro 1991.

[MAAR91] Y. S. MAAREK; Software Library Construction from an IR Perspective; SIGIR Forum, 25(2), 1991.

[MATS87] K. MATSUMURA, K. FURUYA, A YAMASHIRO, T. OBI; Trend Toward Reusable Module Component: Design and Coding Technique 50SM; 10th International Conference on Software Engineering; pp. 45-52; 1987.

[MCCA76] T. J. McCABE; A Complexity Measure; IEEE Transactions on Software Engineering, Vol. SE-2, N. 4, Dezembro, 1976.

[MCCA79] J. A. McCALL; An Introduction to Software Quality Metrics and Software Quality Management; J. D. Cooper e M. J. Fischer (editores); Petrocelli, pp. 127-142; 1979.

[METC91] M. METCALF, J. REID; Whither Fortran?; Fortran Forum - SIGPLAN Special Interest Publication on Fortran; ACM Press; Vol.10 N.2 (S.N. 27), Julho 1991.

[MEYE87] B. MEYER; Reusability: The Case for Object Oriented Design; IEEE Software vol. 4(2); Março, 1987.

[MEYE88] B. MEYER; Object Oriented Software Construction; New York, NY: Prentice Hall; 1988.

[MSFO89a] MICROSOFT CORPORATION; Quick Reference Guide - Microsoft FORTRAN V5.0; Part No. 06385; 1989.

[MSFO89b] MICROSOFT CORPORATION; Reference - Microsoft FORTRAN V5.0; Part No. 06383; 1989.

[MSVC93a] MICROSOFT CORPORATION; Microsoft Visual C++ 1.0 - Programmer's Guides (C++ Tutorial, Class Library e Programming Techniques); Doc. No. DB29682-0193; 1993.

[MSVC93b] MICROSOFT CORPORATION; Microsoft Visual C++ 1.0 - User's Guides (Visual Workbench e App Studio); Doc. No. DB296821-0193; 1993.

[NEIG89] J. M. NEIGHBORS; Draco: A Method for Engineering Reusable Software Systems; em Software Reusability: Vol.I, Concepts and Models; T.J. BIGGERSTAFF e A.J. PERLIS (editores), Addison-Wesley Publishing Company; 1989.

[NUSE93] R. E. NUSENOFF, D. C. BUNDE; A Guidebook and a Spreadsheet Tool for a Corporate Metrics Program; J. Systems Software; 23:245-255; 1993.

[O'SHE86] T. O'SHEA et all; The Learnability of Object Oriented Programming Systems; Proceedings of the ACM Conference on Object Oriented Systems, Languages and Applications; pp. 502-504; 1986.

[OSTE92] E. OSTERTAG, J. HENDLE, R. PRIETO-DIAZ, C. BRAUN; Computing Similarity in a Reuse Library System: An AI-Based Approach; ACM Transactions on Software Engineering and Methodology Vol. 1 n° 3 pp. 205-228; Julho 1992.

[PATE91] S. PATEL, W. SHU, R. BAXTER, B. SAYRS, S. SHERMAN; A top-Down Software Reuse Support Environment; Proc. Fourth Annual Workshop on Software Reuse; Novembro, 1991.

[PATE92] S. PATEL, A. STEIN, P. COHEN, R. BAXTER S. SHERMAN; Certification of Reusable Software Components; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[PETE91] A.S. PETERSON; Comings to Terms with Software Reuse Terminology: A Model-Based Approach; ACM Software Engineering Notes Vol. 16 n ° 2 pp. 45-51; Abril 1991.

[PINS90] L.J. PINSON; R.S. WIENER; Applications of Object-Oriented Programming; Addison-Wesley Publishing Company; 1990.

[POOR88] J. H. POORE; Derivation of Local Software Quality Metrics; Software - Practice and Experience, Vol. 18(11), 1017-1027, Novembro, 1988.

[POUL92] J. S. POULIN; Measuring Reuse; Proc. Fifth Annual Workshop on Software Reuse; Outubro, 1992.

[PRES92] R. S. PRESSMAN; Software Engineering - A Practioner's Approach; McGraw-Hill International Editions; 1992.

[PRIE87a] R. PRIETRO-DIAZ, P. FREEMAN; Classifiying Software for Reusability; IEEE Software, Vol. 4, N. 1, Janeiro 1987.

[PRIE87b] R. PRIETO-DIAZ; Domain Analysis for Reusability; IEEE 11th Compsac, pp. 23-29, Outubro 1987.

[PRIE89] R. PRIETRO-DIAZ; Classification of Reusable Modules; em Software Reusability: Vol.I, Concepts and Models; T.J. BIGGERSTAFF e A.J. PERLIS (editores), Addison-Wesley Publishing Company; 1989.

[PRIE90] R. PRIETO-DIAZ; Domain Analysis: An Introduction; ACM Software Engineering Notes Vol. 15 n° 2 pp. 47-53; Abril 1990.

[PRIE91a] R. PRIETO-DIAZ; Making Software Reuse Work: An Implementation Model; ACM Software Engineering Notes Vol 16 n° 3 pp. 61-68; Julho 1991.

[PRIE91b] R. PRIET-DIAZ; Implementing Faceted Classification for Software Reuse; Communication of the ACM, Vol. 34 n° 5 pp. 89-97; Maio 1991.

[PRIE93] R. PRIET-DIAZ; Status Report: Software Reusability; IEEE Software, Maio 1993.

[RIJS93] D. B. B. RIJSENBRIJ, A. H. BAUER; A Quality System for a Software House; J. Systems Software, 23:211-224; 1993.

[RINE91] D.C. RINE; A Proposed Standard Set of Principle for Object-Oriented Development; ACM Software Engineering Notes Vol. 16 n° 1 pp. 43-49; Janeiro, 1991.

[ROCH83] A.R.C. ROCHA; Um Modelo para Avaliação da Qualidade de Especificações; Tese de Doutorado, PUC-RJ; 1983.

[ROCH87] A.R.C. ROCHA; Análise e Projeto Estruturado de Sistemas; Editora Campus; 1987.

[SELB89] R. W. SELBY; Quantitative Studies of Software Reuse; em Software Reusability: Vol.II, Applications and Experience; T.J. BIGGERSTAFF e A.J. PERLIS (editores), Addison-Wesley Publishing Company; 1989.

[SHLA88] S. SHLAER, S. J. MELLOR; Object-Oriented Systems Analysis: Modeling the World in Data, Prentice Hall, Englewood Cliffs, N.J.,1988.

[SWAN91] K. SWANSON, et. al; The Application Software Factory: Applying Total Quality Techniques to Systems Development; MIS Quartely; pp. 567-579; Dezembro 1991.

[TARU88] H. TARUMI, K. AGUSA, Y. OHNO; A Programming Environment Supporting Reuse of Object-Oriented Software; Proceedings of the 10th International Conference on Software Engineering; pp. 265-273; 1988.

[TESL86] L. TESLER; Object Oriented Languages: Programming Experiences; Revista BYTE; pp. 67-71; Agosto 1986.

[TEWA92] R. TEWARI; Empirical Investigation of Software Reuse in Object-Oriented Systems; Proc. Fifth Annual Workshop on Software Reuse; Outubro 1992.

[TRAC88a] W. TRACZ; Software Reuse Myths; ACM Software Engineering Notes, Vol. 13 n° 1 pp. 17-21; Janeiro 1988.

[TRAC88b] W. TRACZ; Software Reuse Maxims; ACM Software Engineering Notes Vol. 13 n° 4 pp. 28-31; Outubro 1988.

[TRAC90] W. TRACZ; Where Does Reuse Start?; ACM Software Engineering Notes Vol. 15 n° 2 pp. 42-46; Abril 1990.

[TRAC92] W. TRACZ; Domain Analysis Working Group Report - First International Workshop on Software Reusability; ACM Software Engineering Notes Vol. 17 n° 3 pp. 27-31; Julho 1992.

[VAXF84a] DIGITAL EQUIPMENT CORPORATION; Programming in VAX FORTAN - VAX/VMS V4; AA-DO34D-TE, Setembro 1984.

[VAXF84b] DIGITAL EQUIPMENT CORPORATION; VAX FORTAN - User's Guide VAX/VMS V4; AA-DO35D-TE, Setembro 1984.

[WALL92] K. C. WALLNAU; Towards an Extended View of Reuse Libraries; Proc. Fifth Annual Workshop on Software Reuse; Outubro 1992.

[WEID91] BRUCE W. WEIDE, W. F. OGDEN, S. H. ZWEBEN; Reusable Software Components; Advances in Computers, Vol. 33; 1991.

[WERN92a] C. M. L. WERNER; Reutilização de Software no Desenvolvimento de Software Científico; Tese de Doutorado em Engenharia de Sistemas e Computação; COPPE/UFRJ, 1992.

[WERN92b] C. M. L. WERNER, F. A. MATTOS; Um Ambiente para o Desenvolvimento Baseado na Composição de Aplicações; Anais do VI Simpósio Brasileiro de Engenharia de Software; Gramado-RS, 1992.

[WIND92] MICROSOFT; Microsoft Windows - Guia do Usuário; Microsoft Corporation 1992.

[YGLE92] K. P. YGLESYAS; A top-Down Software Reuse Support Environment; Proc. Fifth Annual Workshop on Software Reuse; Outubro 1992.

[YGLE93] K. P. YGLESYAS; Information Reuse Parallels Software Reuse; IBM Systems Journal, Vol. 32, N.4, 1993.

[ZADE73] L.A.ZADEH; Outline of a New Approach to the Analysis of Complex Systems and Decision Processes; IEEE Transactions on Systems, Man, and Cybernetics; Vol. SMC-3, N. 1, Janeiro 1973.

[ZADE84] L.A.ZADEH; Making Computers Think Like People; IEEE Spectrum, Agosto 1984.

[ZADE88] L.A.ZADEH; Fuzzy Logic; IEEE COMPUTER, Abril 1988.