Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

JEOSATAN CARLOS SIMÃO

# UTILIZAÇÃO DA PROGRAMAÇÃO MODULAR VOLTADA PARA LINGUAGEM ORIENTADA A EVENTOS - PROGRESS

Belo Horizonte
2011

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

# UTILIZAÇÃO DA PROGRAMAÇÃO MODULAR VOLTADA PARA LINGUAGEM ORIENTADA A EVENTOS - PROGRESS

por

Jeosatan Carlos Simão

Monografia de Final de Curso

Prof. Roberto da Silva Bigonha
Orientador

Belo Horizonte
2011

JEOSATAN CARLOS SIMÃO

# UTILIZAÇÃO DA PROGRAMAÇÃO MODULAR VOLTADA PARA LINGUAGEM ORIENTADA A EVENTOS - PROGRESS

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do certificado de Especialista em Informática.

Área de concentração: Engenharia de Software

Orientador: Prof. Roberto da Silva Bigonha
Universidade Federal de Minas Gerais

Belo Horizonte

2011

*“A grande conquista é o resultado de pequenas vitórias que passam despercebidas.”*

(Paulo Coelho)

## RESUMO

O principal objetivo do presente trabalho é apresentar a fundamentação teórica para os profissionais de sistemas e empresas que utilizam produtos de ERP - Enterprise Resource Planning, com código fechado baseado na linguagem de desenvolvimento Progress, para que possam seguir como orientação as técnicas de modularidade e engenharia de software na realização das customizações introduzidas no sistema. São tratados alguns conceitos básicos sobre programação modular juntamente com as melhores práticas de engenharia de software que podem ser aplicadas a linguagem Progress na análise e desenvolvimento de software. A utilização dessas técnicas proporciona uma significativa diminuição de custo no desenvolvimento do software além de facilitar e padronizar a manutenção. A customização aliada a um desenvolvimento modular oferece agilidade, produtividade e qualidade dos sistemas específicos da organização, deixando-os mais competitivo. No fim do trabalho, são apresentados os resultados obtidos com o estudo.



**Palavras-chave:**. ERP, Progress, Modularidade, Engenharia de software.

## ABSTRACT

The main aim of this paper is to show the theoretical basis of the professional systems and companies that use ERP products - Enterprise Resource Planning, with proprietary code based the Progress language, can follow, as guidelines, the techniques of modularity and software engineer for the execution of introduced customizations in the system. It will treat some basic concepts about modular programming together with the best practices of software engineering that can be used the Progress language in the analysis and development of software. The use of these techniques allows a reliable decrease of costs in the development of software, keeps the maintenance easy. The customization together with modular development offers agility, productivity and quality of the specific systems of the organization, making it more competitive. At the end of this work is shown the results obtained with this study.



**Keywords*:*** ERP, Progress, Modularity, Software Engineering.

## LISTA DE FIGURAS

## LISTA DE QUADROS

## LISTA DE SIGLAS

| | |
|---|---|
| BI | Before-Image |
| DB | Data Base |
| ERP | Enterprise Resource Planning |
| GOF | Gang of Four |
| UPC | User Program Call |

**SUMÁRIO**

# 1 INTRODUÇÃO

O desenvolvimento de software é um processo complexo que requer recursos financeiros e humanos. A customização de sistemas pré-desenvolvidos é mais trabalhosa, pois afeta toda a estrutura do produto e requer técnicas avançadas de desenvolvimento e engenharia de software.

O ERP (Enterprise Resource Planning) Data Sul da Totvs é um sistema baseado na plataforma Progress e direcionado para médias e grandes empresas que buscam ter um maior controle de suas atividades em seus diversos departamentos. Quando adquirido, o produto vem com pacote básico de funcionalidades, o que permite a utilização do sistema por empresas de vários segmentos.

Segundo Costa (2000) o Progress é uma linguagem de programação altamente robusta com um banco de dados relacional de altíssima performance e extrema segurança embutido em um único produto.

As empresas que adquirem o ERP Data Sul vendido pela Totvs necessitam customizar suas atividades específicas, para adequar o sistema à sua regra de negócio. Para a realização dessa customização, é necessário que os todos os envolvidos possuam noções básicas dos princípios de engenharia de software.

A engenharia de software é uma disciplina da computação que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação (SOMMERVILLE, 2003).

## 1.1 Objetivo geral

Apresentar fundamentação teórica para análise e desenvolvimento de customizações para sistemas ERP de código fechado, desenvolvidos na plataforma Progress utilizando técnicas de modularidade e engenharia de Software.

## 1.2 Objetivos específicos

- Conhecer os conceitos de programação modular;
- Apresentar a linguagem de desenvolvimento Progress;
- Descrever o uso da engenharia de software, programação modular x Progress;
- Apresentar resultados obtidos com base nos estudos realizados.

## 1.2 Estrutura da monografia

Esta monografia está organizada da seguinte forma: Seção 2 descreve qualidade de software, por exemplo como a qualidade interna e externa de um sistema será apresentando a opinião de alguns especialistas. A Seção 3 mostra o que é um software modular, destacando os conceitos mais importantes tais como critérios, regras e princípios de uma programação modular. Seção 4 apresenta os benefícios da manutenibilidade na produção de software e as técnicas utilizadas para sua obtenção. Na seção 5 esclarece o que é conectividade e estabilidade de um software juntamente com as técnicas de acoplamento e coesão. Na seção 6 é apresentado a linguagem de desenvolvimento Progress e por ultimo conclui-se este

trabalho com análise de viabilidade da utilização de sistemas modulares desenvolvidos em progress.

# 2 QUALIDADE DE SOFTWARE

Um dos principais objetivos que se deseja ao desenvolver softwares utilizando o conceito de engenharia de software é a qualidade. Pressman (2002) define qualidade de software como conformidade a requisitos funcionais e de desempenho explicitamente declarados, padrões de desenvolvimento explicitamente documentados e características implícitas, que são esperadas em todo software desenvolvido profissionalmente.

Projetar software é tarefa árdua e tem demandado atenção de pesquisadores e profissionais desde os primórdios da produção de software. Myers (1975) descreve que talvez o maior problema enfrentado é a extrema dificuldade de criar sistemas com alto nível de qualidade além do custo de desenvolver e manter sistemas de programação de grande porte.

Ainda hoje, com todas as técnicas de análise, desenvolvimento e engenharia de software cada vez mais aprimorada a situação não é diferente.

O cliente que compra o software deseja um produto que atenda suas especificações, quem o desenvolve necessita que o software tenha qualidade para poder otimizar seus processos, garantir que o produto final atenda as especificações estabelecidas, reduzir custos entre outros.

Pádua (2009) afirma que a qualidade do produto é o seu grau de conformidade com os respectivos requisitos que definem as características e os critérios de aceitação de um produto.

Pressman (2002) ressalta que à medida que a importância dos softwares na sociedade cresce, aumenta também a preocupação da comunidade produtora de software em obter recursos para desenvolver software de forma mais fácil, mais rápida e com custos menores.

De acordo com a NBR ISO/IEC 9126-1:2003, os atributos de qualidade de software é categorizado em seis características (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade), as quais são, por sua vez, subdivididas em subcaracterísticas (FIG. 1). As subcaracterísticas podem ser medidas por meio de fatores externos e internos.

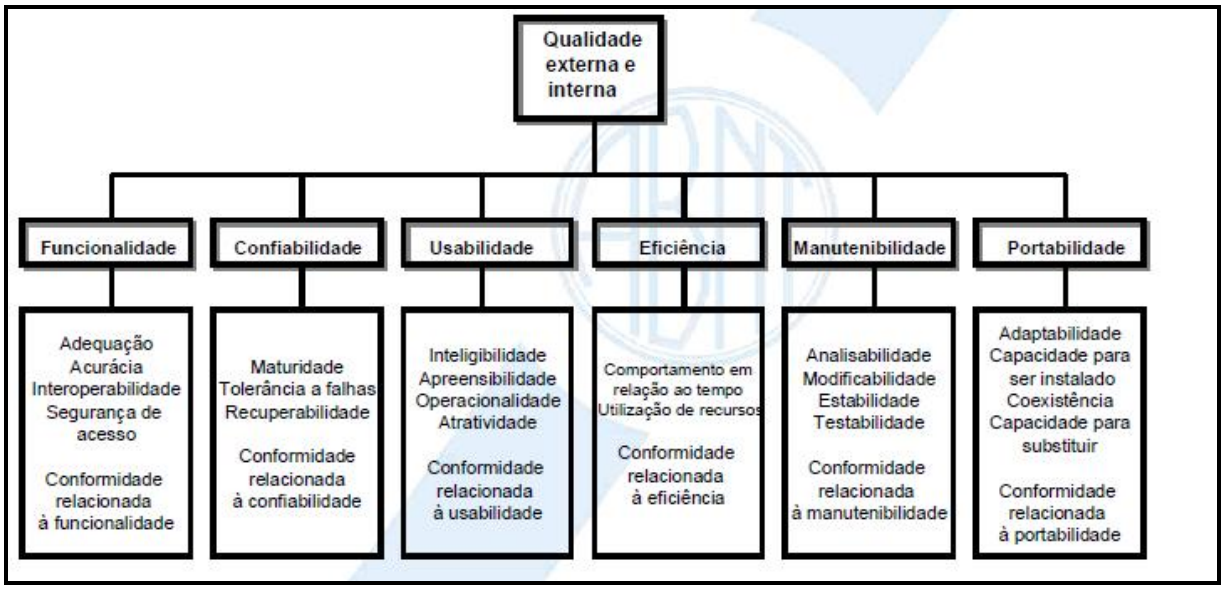


FIG.1 Modelo de qualidade para qualidade externa e interna
Fonte: NBR ISO/IEC 9126-1:2003, p. 7

Embora exista hoje uma disposição dos desenvolvedores de software uma gama de conceitos, tecnologias, metodologias e linguagens de programação poderosas, desenvolver software é ainda muito difícil e caro, um campo fértil a ser explorado e melhorado.

As boas práticas recomendadas por favorecer a construção de software de melhor qualidade dizem respeito à observância dos princípios de modularidade, flexibilidade, alta coesão de métodos e atributos, baixo acoplamento, robustez e aumento do grau de reúso (NOGUEIRA, 2006).

Tanto no projeto, no desenvolvimento ou na utilização de um software, o mesmo deve mostrar-se eficiente e eficaz para assim poder ser considerado um software com projetos modulares, flexíveis e robustos enfim um software de qualidade.

## 2.1 Fatores internos e externos

Um software rápido, viável, fácil de usar, de fácil leitura, modular, estruturado e com todas as características da engenharia de software pode ser considerado um sistema de qualidade.

Podemos dividir as características de qualidade em fatores externos e internos de qualidade. Os fatores que podem ser detectados pelos usuários são identificados como fatores externos.

Meyer (1997) relata que "usuários" não são apenas as pessoas que realmente interagem com o produto final, mas também aqueles que compram o software ou contratam seu desenvolvimento, como por exemplo um executivo de uma companhia.

Dessa forma, uma propriedade, como a facilidade com que o software pode ser adaptado às mudanças de especificações enquadra-se na categoria de fatores externos, embora possa não ser de interesse imediato para os usuários finais. Os principais fatores externos de qualidade de software são:

- Correção: a capacidade de o software realizar as tarefas como foram definidas em sua especificação de requisitos. Certamente, esse é o primeiro aspecto a ser observado em um software.
- Robustez: um software pode ser considerado robusto se realiza suas tarefas de forma correta mesmo quando submetido a condições anormais.

- Extensibilidade: característica de um software poder ser facilmente adaptado a inclusões e alterações de requisitos. A vida de um software é longa, e ao longo de sua existência, alterações ou novos requisitos são inevitáveis.
- Reusabilidade: característica de um software poder ser reutilizado ao todo ou em parte por outros softwares. Na produção de software, a possibilidade de se reutilizar elementos já construídos facilita o projeto e o seu desenvolvimento, o que torna o produto mais confiável com geração de impacto principalmente no seu custo.
- Compatibilidade: facilidade de se combinar o software com outros softwares. Essa característica é importante porque raramente um software é construído sem interação com outros softwares.
- Eficiência: refere-se ao bom uso que o software faz dos recursos de hardware, tais como memória e processadores.
- Portabilidade: é a facilidade de se utilizar o software em diversos ambientes de hardware e software.
- Verificabilidade: a facilidade de se preparar rotinas para se verificar a conformidade do software com os seus requisitos.
- Integridade: é uma característica relacionada à segurança de dados, programas e documentos. Integridade é a habilidade de proteger tais componentes contra acessos não autorizados.
- Usabilidade: é a facilidade de uso de um software. Visa aperfeiçoar a interação do homem com a máquina. O projeto de interfaces de usuário é uma questão fundamental na produção de software e deve ser realizado de forma a garantir questões como produtividade, facilidade de uso e aprendizado do software pelos usuários.

A NBR ISO/IEC 9126-1:2003 define a qualidade apresentada quando o software é executado como qualidade externa. Esse tipo de qualidade é tipicamente medido e avaliado enquanto está testa-se o software num ambiente simulado, com dados simulados e usando métricas que possam ser analisadas.

Perceptíveis apenas aos profissionais de informática que têm acesso ao software de texto real qualidades tais como ser modular, ou legível, são fatores internos. Meyer

(1997) defende que os fatores internos são a chave para a realização dos fatores externos é que para os usuários desfrutar das qualidades visíveis (externas), os designers e implementadores devem aplicar técnicas internas de maneira a garantir as qualidades internas.

A NBR ISO/IEC 9126-1:2003 diz que a qualidade interna é medida e avaliada com relação aos requisitos de qualidade, sendo a totalidade das características do produto de software do ponto de vista interno. Detalhes da qualidade do produto de software podem ser melhorados durante a implementação do código, revisão e teste.

Os fatores internos de qualidade é o meio para se obter os fatores externos, mas são ambos que vão dar o titulo de sistema de qualidade ao produto final. Não é possível afirmar que um software que atende os requisitos externos mas não foi projetado e desenvolvido seguindo os padrões de qualidade interna e engenharia de software seja um software com qualidade total e muito menos um software que atenda aos requisitos internos mas não se mostra eficaz ao usuário final seja avaliado como um sistema com qualidade total.

## 2.2 Qualidade de uso

A qualidade de uso consiste na qualidade do software no momento de sua execução, sendo perceptível principalmente pelo cliente final.

A NBR ISO/IEC 9126-1:2003 diz que é a visão da qualidade sob a perspectiva do usuário. A obtenção de qualidade em uso é dependente da obtenção da necessária qualidade externa, a qual, por sua vez, é dependente da obtenção da necessária qualidade interna.

Para que a capacidade do produto de software permita que usuários especificados atinjam metas especificadas com eficácia, produtividade, segurança e satisfação em

contextos de uso especificados seja atingida. A NBR ISO/IEC 9126-1:2003 criou o modelo para qualidade de uso, que está demonstrado na figura abaixo.

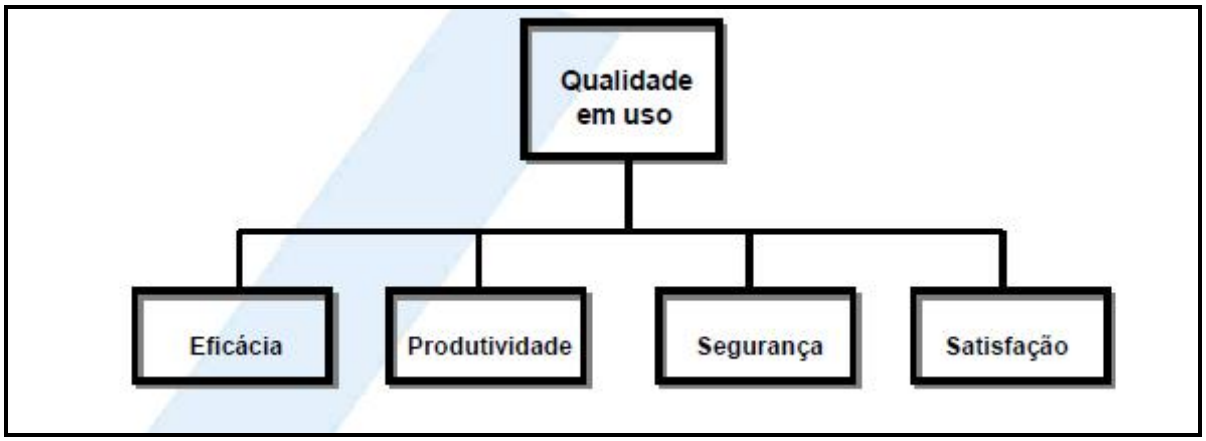


FIG.2 Modelo de qualidade de uso
Fonte: NBR ISO/IEC 9126-1:2003, p. 11

O quadro abaixo foi elaborado com a descrição de cada item que compõem a qualidade de uso baseado na NBR ISO/IEC 9126-1:2003.

QUADRO 1

Características dos critérios da qualidade de uso

| Critério | Descrição |
|---|---|
| Eficácia | Capacidade do produto de software de permitir que usuários atinjam metas especificadas com acurácia e completitude, em um contexto de uso especificado. |
| Produtividade | Capacidade do produto de software de permitir que seus usuários empreguem quantidade apropriada de recursos em relação à eficácia obtida, em um contexto de uso especificado. |
| Segurança | Capacidade do produto de software de |

| | |
|---|---|
| | apresentar níveis aceitáveis de riscos de danos a pessoas, negócios, software, propriedades ou ao ambiente, em um contexto de uso especificado. |
| Satisfação | Capacidade do produto de software de satisfazer usuários, em um contexto de uso especificado. |

Fonte: Elaboração gráfica própria com informações extraídas de (NBR ISO/IEC 9126-1:2003, p.12, tradução nossa)

## 2.3 Padrão de projeto

O uso de padrões de projeto pode contribuir na produção de software de alta qualidade, principalmente pela possibilidade de reúso de soluções de projeto e programação. Padrões de software podem se referir a diferentes níveis de abstração no desenvolvimento de software. Existem padrões para a solução de problemas de análise, projeto e programação de software.

Uma especificação de requisitos deve ser completa, correta, precisa e verificável, pois nela deve estar definido o objetivo do software, os requisitos que ele deve atender. Contudo, a fase determinante da qualidade do software é o projeto, pois nesta fase define-se como o software será construído. O projeto deve ser criterioso, atender aos requisitos especificados e modelar o sistema de forma a obter uma estrutura flexível (PÁDUA, 2009).

Cada padrão de projeto descreve um problema em nosso ambiente e o núcleo da sua solução. Dessa forma podemos utilizá-lo milhões de vezes, sem nunca utilizá-lo do mesmo modo.

> Cada padrão de projeto sistematicamente nomeia, explica e avalia um aspecto de projeto importante e recorrente em sistemas orientados a objetos, capturando a experiência de projeto de uma forma que as pessoas possam utilizá-la efetivamente. (GAMMA et alii, 2000, p. 20).

Nomear é referenciar o que pode ser usado para descrever um problema de projeto, suas soluções e conseqüências em uma ou duas palavras. O nome de um padrão possibilita uma melhor comunicação e a construção de projetos em nível mais alto de abstração, o problema descreve em que situação aplicar o padrão, explicando o problema em seu contexto e podendo incluir, às vezes, uma lista de condições que devem ser satisfeitas para que faça sentido aplicá-lo.

GAMMA et alii (2000) afirmam ainda que a solução descreve os elementos, classes e objetos, que compõem o padrão de projeto, seus relacionamentos, suas responsabilidades e colaborações mas lembram que a solução não descreve um projeto concreto ou uma implementação em particular e sim fornece uma descrição abstrata de um problema de projeto, e de como um arranjo geral de elementos o soluciona.

O referido autor ressalta também que as conseqüências são os resultados e análises das vantagens e desvantagens da aplicação do padrão, ou seja, são críticas para a avaliação das alternativas de projetos e para a compreensão dos custos e benefícios da aplicação do padrão.

Os padrões desenvolvidos nos últimos anos são adições incrementais aos já existentes incluindo as técnicas de uso geral, nessa visão a principal contribuição é o padrão de software em si e não a idéia de utilização de um padrão (MEYER,1997).

Segundo GAMMA et alii (2000), a utilização dos padrões de projeto trazem muitos benefícios à comunidade de software:

- Estabelecem uma terminologia comum para melhorar a comunicação entre equipes;
- Permitem o aprendizado com a experiência de outros sem ter que reinventar soluções para problemas recorrentes;

- Facilitam o repasse do conhecimento entre os desenvolvedores experientes;
- Permitem o reúso de soluções existentes de alta qualidade em problema normalmente recorrente;
- Facilitam a criação de arcabouços contendo padrões de projeto já implementados, possibilitando o reúso dos padrões;
- Propiciam melhor nível de modularidade;
- Propiciam estabilidade e evolução do código;
- Favorecem o desacoplamento entre áreas de responsabilidade de modo que as mudanças em uma área não ocasionem mudanças nas outras;
- Aumenta produtividade.

Uma coleção de padrões muito utilizadas é a proposta por Gamma et alii, chamados de padrões GOF (Gang of Four), que classifica vários padrões de acordo com seu escopo. Os princípios podem ser de origem estrutural, de criação ou comportamental Ferreira (2006). No quadro abaixo GAMMA et alii (2000) mostra a divisão dos padrões e sua classificação (QUADRO 2).

QUADRO 2

Classificação dos Padrões GOF

| | | **Propósito** | | |
|---|---|---|---|---|
| | | **De criação** | **Estrutural** | **Comportamental** |
| **Escopo** | **Classe** | Factory Method | Class Adapter | Interpreter<br>Template Method |
| | **Objeto** | Abstract Factory<br>Builder<br>Prototype<br>Singleton | Object Adapter<br>Bridge<br>Composite<br>Decorator<br>Façade<br>Flyweight<br>Proxy | Chain of Responsability<br>Command<br>Iterator<br>Mediator<br>Memento<br>Observer<br>State<br>Strategy<br>Visitor |

Fonte: Gamma, 2000, p.26

Ferreira (2006) salienta que um padrão possui vários elementos (nome, descrição do problema, solução dada e as conseqüências de sua utilização), entretanto, a aplicação do padrão não é imediata, sendo necessário analisar cuidadosamente se seu uso é adequado ao problema que se quer solucionar e quais as conseqüências.

Em termos gerais, a utilização dos padrões de projeto é fundamental para que se obtenha um software que atenda as premissas da engenharia de software, uma vez que permite uma normalização e documentação dos processos a serem utilizados no desenvolvimento do produto.

# 3 SOFTWARE MODULAR

A idéia de software modular consiste em separar o sistema em pequenos pedaços de forma a simplificar o seu desenvolvimento e sua manutenção. Com essa técnica é possível obter um maior desempenho de toda a equipe envolvida na confecção do produto alem de garantir a sua continuidade.

Conforme Staa (2000) a programação modular é um instrumento que viabiliza atingir os princípios de qualidade e tem por objetivo servir como base para o desenvolvimento de software de forma organizada e com maior reusabilidade, de forma a assegurar a manutenibilidade e qualidade do programa.

A modularidade possui como principal característica a independência entre os componentes de software viabilizando a obtenção da reusabilidade.

Myers (1975) descreve a modularidade como resposta a realização de alterações no software de forma mais fácil e com menor custo pois possibilita que a modificação de um determinado módulo afetem um número pequeno de outros módulos reduzindo a sua complexidade.

De acordo com Bigonha (2001), um módulo é um componente de software que possa ser compilado separadamente. A forma como essa articulação é realizada determina a flexibilidade do sistema.

Staa (2000) pontua as vantagens da programação modular:

- Facilita a distribuição do trabalho ao dividir o problema;
- Permite criar um acervo de ativos de software trazendo benefícios para a organização;

- Torna gerenciável o processo de desenvolvimento por permitir que se criem linhas de base dos módulos já aprovados e o desenvolvimento incremental de programas;
- Por meio da criação de versões sucessivas e cada vez mais completas contempla o desenvolvimento incremental de programas;
- Possibilita a criação de versões sucessivas e cada vez mais completas podendo deixar o aprimoramento do desempenho para uma época mais oportuna;
- Reduz o tempo de compilação, pois somente os módulos afetados pelas alterações deverão ser recompilados;
- Reduz o custo de desenvolvimento dos sistemas, ao criar projetos que sejam compostos por componentes reutilizáveis e já aprovados;
- Favorece a extensibilidade do sistema via o critério da continuidade.

Meyer (1997) ressalta que a modularidade ajuda os designers a produzir sistemas de software feito de elementos autônomos ligados por uma estrutura simples e coerente. Segundo o autor, essa estrutura simples permite que as vantagens da modularidade sejam atingidas e o software adquira um conceito elevado de qualidade e competitividade no mercado.

## 3.1 Critérios de modularidades

De acordo com Meyer (1997) um sistema que possa ser avaliado como modular deve satisfazer a cinco requisitos fundamentais:

- Decomposibilidade: esse critério permite a construção de software a partir da decomposição do problema a ser resolvido, dividindo-se o problema em subproblemas. A figura abaixo (FIG. 2) mostra a decomposição de um sistema em subsistemas e a partir daí o trabalho será divido nesses grupos.

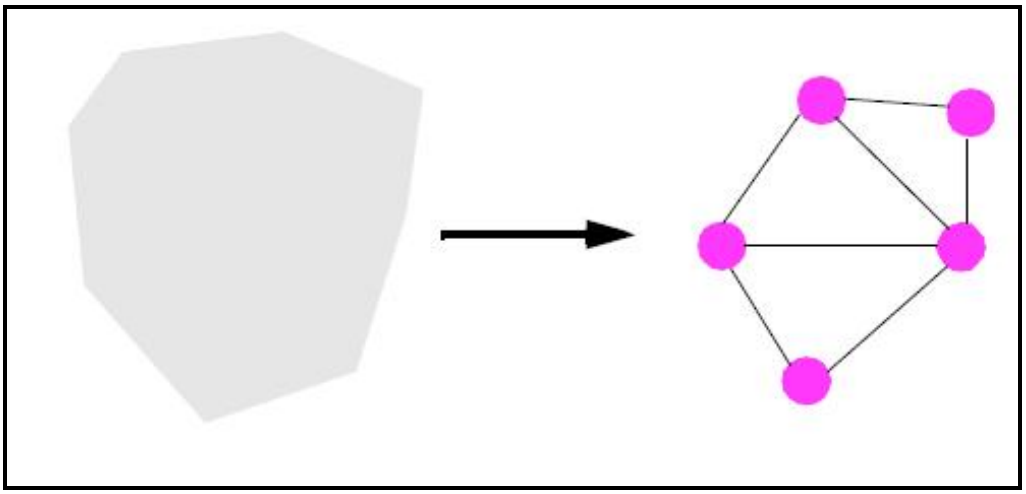

FIG.3 Modelo de decomposibilidade
Fonte: Meyer, 1997, p. 40

- Composibilidade: permite que os elementos possam ser livremente combinados entre si para produzir novos sistemas, possivelmente em um ambiente completamente diferente daquele em que foram inicialmente desenvolvidos. A figura abaixo (FIG. 3) mostra que elementos do contexto para o qual foram originalmente concebidos, podem ser utilizados novamente em diferentes contextos.

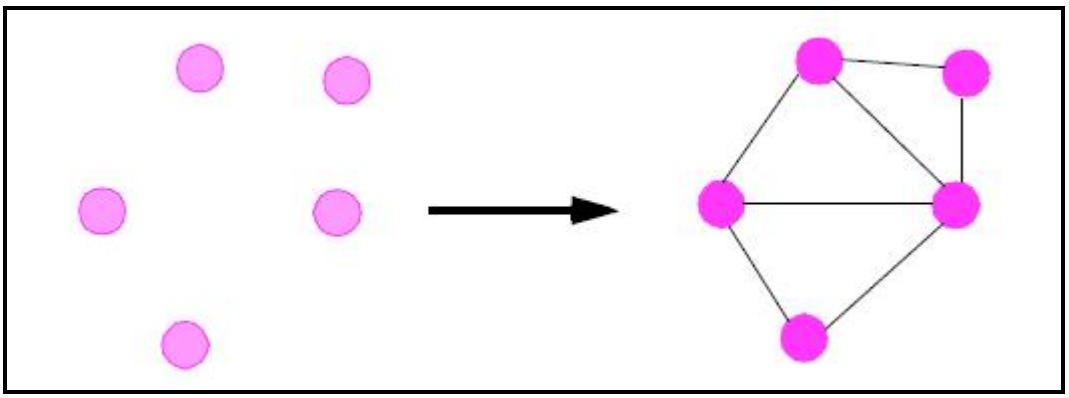

FIG.4 Modelo de composibilidade
Fonte: Meyer, 1997, p. 42

- Inteligibilidade: em um método que segue esse critério, é possível entender um módulo sem a necessidade de recorrer aos demais módulos, é a independência dos módulos. A figura a seguir (FIG. 4) demonstra que um leitor de código pode compreender seus elementos separadamente.

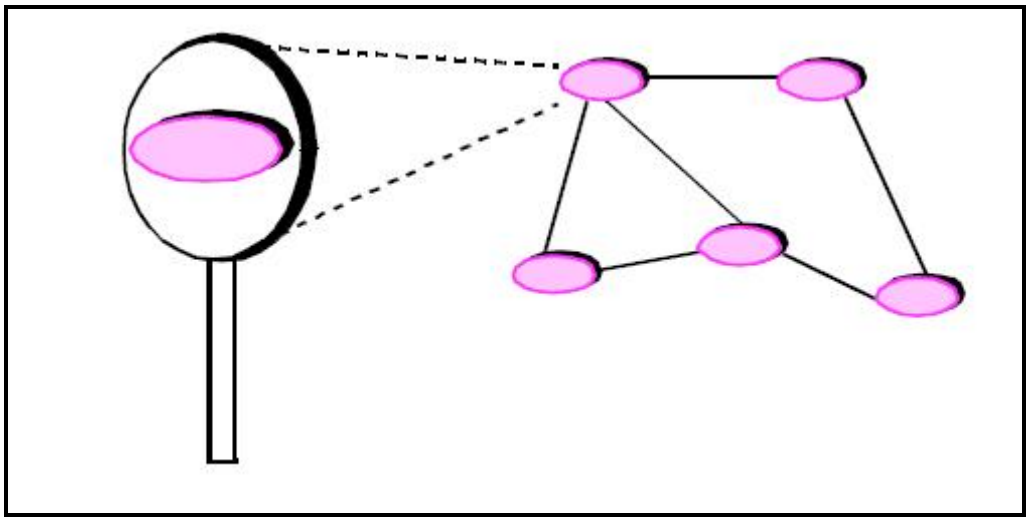

FIG.5 Modelo de inteligibilidade
Fonte: Meyer, 1997, p. 43

- Continuidade: possibilita que quando ocorrer alguma alteração de especificação em um módulo resultará em poucas ou nenhuma alteração nos demais módulos.
- Proteção: determina que a ocorrência de um erro dentro de um módulo em momento de execução, deve gerar pouca ou nenhuma propagação de erros para os demais módulos, devendo ficar contida dentro do módulo originário. O modelo abaixo (FIG. 5) mostra a contenção da propagação do erro dentro do módulo em que ocorreu o evento.

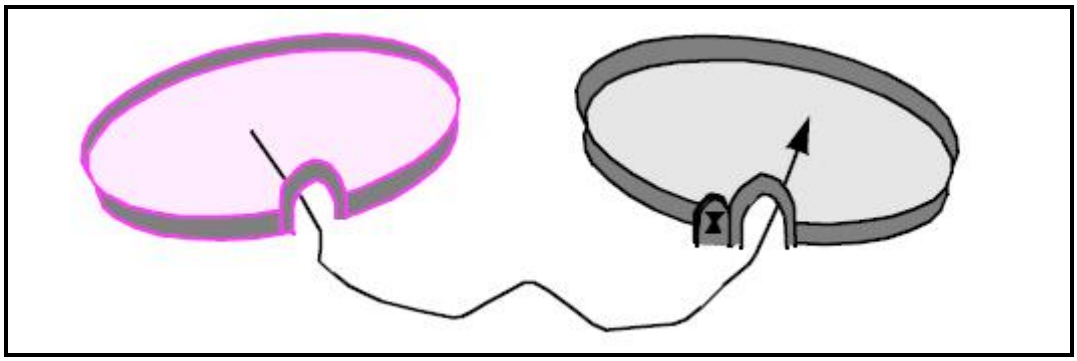

FIG.6 Modelo de proteção
Fonte: Meyer, 1997, p. 46

A utilização dos critérios de modularidade permite produzir projetos reutilizáveis, manuteníveis, extensíveis e robustos.

## 3.2 Regras da modularidade

Meyer (1997) apresentou cinco princípios de modularidade que devem ser seguidos nos sistemas modulares visando uma estrutura mais flexível e com menor dependência entre os módulos.

- Mapeamento direto: determina que para cada problema a ser solucionado deve ser definido um modelo de problema e para cada modelo de problema deve existir um correspondente modelo de solução.
- Pouca interface: a estrutura do sistema deve conter poucas interfaces entre os módulos que o compõe, quanto menos atravessadores existir entre um módulo e outro melhor.
- Interface pequena: a comunicação entre módulos pode ocorrer em uma enorme variedade de maneiras, contudo, o principal é que se ocorre comunicação ela devem ela deve conter a menor quantidade possível de informação pois favorece a extensibilidade do sistema, a figura 6 demonstra um modelo de interface pequena.

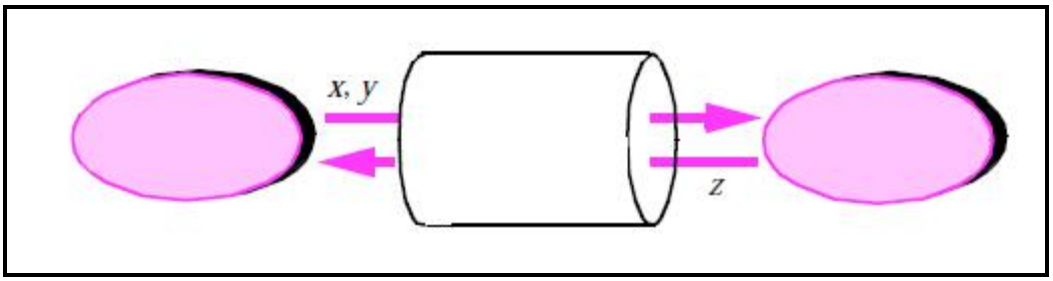


FIG.7 Modelo de interface pequena
Fonte: Meyer, 1997, p. 48

- Interface explícita: sempre que ocorrer uma comunicação entre dois módulos essa comunicação deve ser explicitada em um dos módulos ou em ambos. Se as conexões são claras, fica mais fácil reutilizar e manter os módulos. Abaixo um modelo de comunicação explícita.

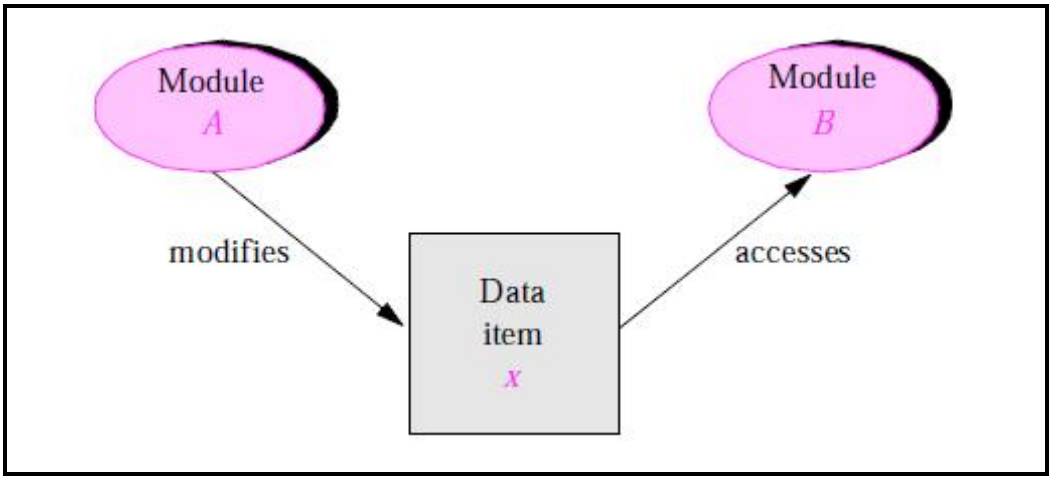


FIG.8 Modelo de comunicação explicita
Fonte: Meyer, 1997, p. 50

- Ocultação de informação: as informações de uso geral de um módulo devem ser disponibilizada para todo o sistema, já as informações próprias do módulo devem ser ocultas para uso próprio conforme modelo abaixo. Ferreira (2006) afirma que um módulo deve ser conhecido apenas por sua interface, que seriam as informações publicas, já as demais informações são privadas.

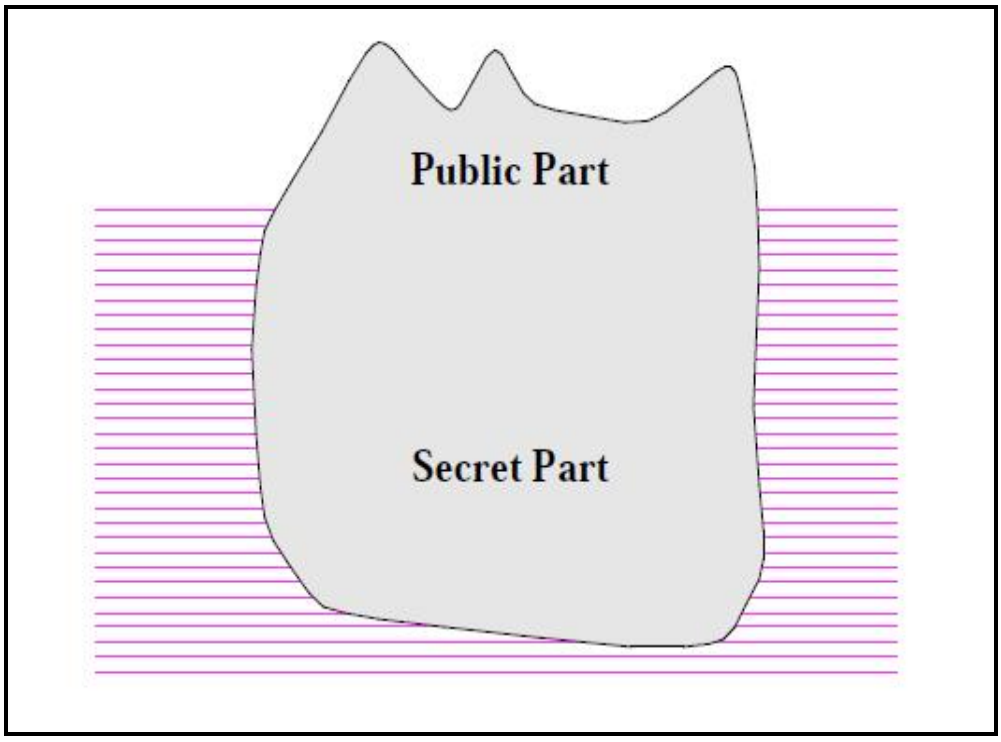


FIG.9 Modelo de ocultação de informação
Fonte: Meyer, 1997, p. 51

Em programas bem modularizados, cada módulo corresponde a uma abstração do que é relevante no contexto do problema. Com uma abstração bem definida do módulo e seguindo os critérios de princípios de modularidade consegue-se obter uma comunicação bem definida entres todos os módulos do sistema.

Staa (2000) ratifica que a interface entre os módulos deve ser bem definida e respeitada pelos programadores, considerando-se interface como o mecanismo por meio do qual se realiza a troca de dados, de comandos e de eventos entre os elementos de um ou mais programas, módulos, classes ou funções.

## 3.2 Princípios da modularidade

Os critérios e as regras da modularidade são os meios para conseguir obter os seus princípios. Meyer (1997) define os princípios da modularidade em:

- Unidade linguística: expressa que o formalismo utilizado para descrever software em vários níveis (especificações, projetos, implementações) devem apoiar a vista da modularidade, normalmente os módulos devem corresponder a unidades sintáticas na língua utilizada. A linguagem utilizada no projeto deve oferecer recursos para a realização das abstrações do domínio do sistema.
- O princípio de documentação: devemos eliminar o sentido da ocultação da informação. Os envolvidos no desenvolvimento de um módulo devem se esforçar para tornar toda a informação sobre o que ele deve realizar ou retornar dentro do próprio módulo. A justificativa mais plausível para o princípio da documentação é o critério de inteligibilidade modular, mas mais importante é o papel desse princípio para ajudar a satisfazer o critério de continuidade.
- Acesso uniforme: decorrente do critério de continuidade, pode-se entender como um caso especial de ocultação de informação. Todos os serviços oferecidos por um componente devem estar disponíveis através de uma notação uniforme, que não informa se são implementados através de armazenamento ou por meio de computação.
- O princípio de aberto fechado: princípio básico que define que os módulos devem ser abertos e fechados, de modo que módulos de software devem ser abertas para extensão, mas fechadas para modificação. Novas funcionalidades devem ser adicionadas pelo polimorfismo, reúso ou qualquer outra técnica que não altere o código do elemento já existente. É importante ressaltar que o fechamento deve ser feito quando a estrutura for considerado razoavelmente estável para que nao seja necessario reabertura para eventuais modificações.
- A escolha individual: um sistema de software sempre deve suportar um conjunto de alternativas, e apenas um módulo no sistema deve saber sua lista completa de opções.

# 4 MANUTENIBILIDADE

Manutenibilidade pode ser considerada como a facilidade em se manter um sistema em uso continuamente.adaptando as novas funcionalidades e alterações de escopo.

A NBR ISO/IEC 9126-1:2003 afirma que manutenibilidade é a capacidade do produto de software de ser modificado. As modificações podem incluir correções, melhorias ou adaptações do software devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais.

Ferreira (2006) descreve que se o produto for mal construído, a sua manutenção é gravemente prejudicada, ainda que problemas nos demais fatores, como remanejamento de pessoal e qualidade da documentação, sejam superados.

## 4.1 Reúso

Desde a invenção do computador, o desenvolvimento de software era visto como atividade que sempre privilegiou soluções sob medida, nos quais componentes de hardware e software são projetados e construídos especificamente para determinada aplicação.

Enxergava-se o reúso de componentes como um fator de risco a ser evitado por serem complexos, componentes de prateleira (caixa preta), e por normalmente adicionar novos requisitos ao projeto.

Por outro lado, há fatores econômicos e mercadológicos que impõem mudanças de estratégia tais como alta demanda e prazos cada vez menores além dos sistemas terem de ser mais fáceis de usar e confiáveis.

Meyer (1997) salienta que a reusabilidade é a propriedade de um componente de software poder ser utilizado em novas aplicações. Essa técnica facilita o projeto e o seu desenvolvimento, quanto menor o nível de dependência de um módulo maior a facilidade de reutilizá-lo, o que é a base para manutenibilidade eficaz.

O recurso de polimorfismo impacta diretamente em maior reusabilidade do sistema. Quando bem empregado o reúso permite o mínimo de acoplamento, o que contribui significativamente para a redução do custo do software.

Reúso significa reutilizar mais do que simplesmente o código, podendo fazer reutilização de requisitos, análise, projeto, planejamento de testes, interfaces de usuários e arquiteturas. Assim, praticamente todos os componentes do ciclo de vida da engenharia de software podem ser encapsulados como objetos reusáveis (YOURDON, 1999).

Meyer (1997) assegura que um software mais reutilizável pode-se esperar melhorias nas seguintes frentes:

- Pontualidade;
- Diminuição do esforço de manutenção;
- Confiabilidade;
- Eficiência;
- Coerência;
- Investimento;

Ao se criar catálogos de componentes de software de tal forma que para construir um novo sistema, possa combiná-los de forma apropriada em vez de reinventá-los pode diminuir significativamente o tempo para o desenvolvimento de novas aplicações, bem como a quantidade de código a ser implementado, na medida em

que parte das responsabilidades das novas aplicações são atribuídas a artefatos de software reutilizados. A adoção desta prática no desenvolvimento de software, seja em conjunto ou isoladamente, são fundamentais para a redução dos custos na produção de um sistema além de elevar o nível de confiabilidade.

Sendo bem empregada essa técnica fica claro o reaproveitamento e a estabilidade oferecida ao sistema. O pensamento dos analistas fica focado no comportamento módulo como um todo e não nos detalhes de implementação que por sua vez acaba sendo mais acelerado por existir componentes já prontos que serão apenas reutilizados.

### 4.3 Polimorfismo

O polimorfismo é a habilidade de se esconder implementações distintas atrás de interfaces comuns, tornando os softwares menos sensíveis a alterações o que os tornam mais extensíveis (DEITEL, 2001). Habitualmente o polimorfismo está associado à idéia de que entidades do programa possam ter mais de uma forma ou implementação.

Varejão (2004) pondera que algumas linguagens, chamadas monomórficas, possuem um sistema de tipos no qual variáveis, constantes e subprogramas devem ser definidos com um tipo específico fixo. Embora traga alguns benefícios, este sistema não favorece flexibilidade para se implementar algoritmos e estruturas de dados genéricos e conseqüentemente, não favorecem o reúso, já o polimorfismo se refere à possibilidade de criar código capaz de operar sobre valores de tipos diferentes.

Tipos de dados polimórficos são aqueles cujas operações se aplicam a valores tipos diferentes. Subprogramas polimórficos são aqueles cujos parâmetros e tipo de retorno, se for função, podem assumir valores de mais de um tipo.

Meyer (1997) também defende que a metodologia polimórfica significa a capacidade de assumir diversas formas. Uma entidade variável ou elemento de estrutura de dados, que terá a capacidade, em tempo de execução, de denotar objetos de diferentes tipos, todas controladas por a declaração estática. A seguir é demonstrado o modelo que Meyer usou para mostrar um vetor polimórfico (FIG. 9):

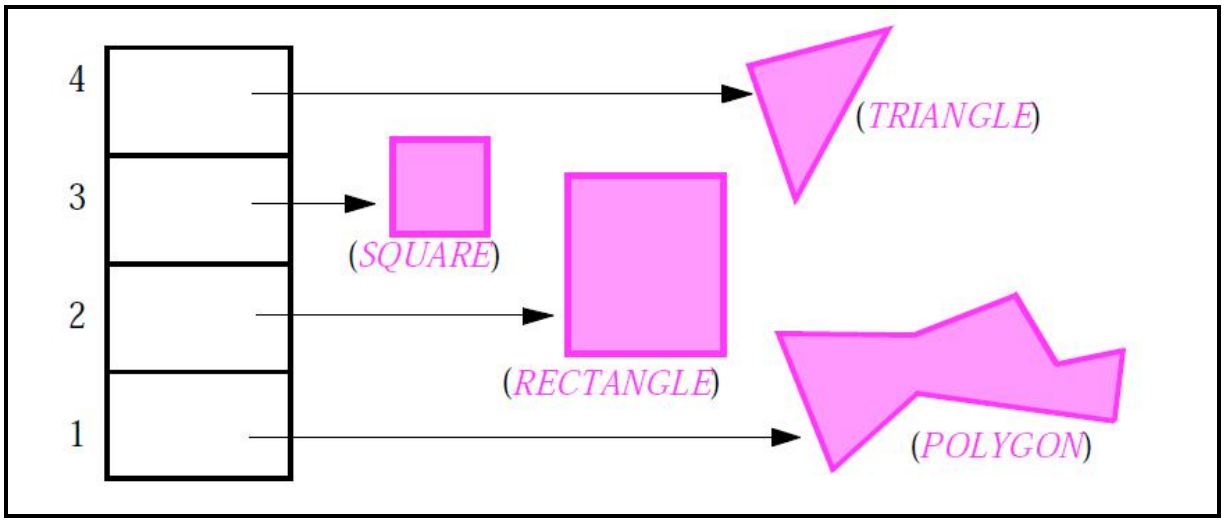


FIG.10 Modelo de vetor polimórfico
Fonte: Meyer, 1997, p. 471

Enfim, polimorfismo fornece flexibilidade ao código em casos em que o tipo do objeto não é previamente conhecido.

## 4.4 Componentes

O desenvolvimento de software baseado em componentes é uma abordagem que permite simultaneamente reduzir custos no desenvolvimento, diminuir o período de implementação necessário e aumentar a qualidade.

Meyer (1997) disse que o desenvolvimento por componentes é a divisão entre especificação do componente e a implementação. A especificação define o comportamento observável externo, como o componente deve funcionar, o que se

espera dele e como ele ira interagir com outros sistemas. A implementação fornece um modelo concreto para instanciação, em diferentes ambientes de acordo com a especificação do componente.

Um conjunto de um ou mais módulos, formando um todo coerente e implementando uma funcionalidade bem definida, como por exemplo um conjunto de funções, uma classe, um tipo abstrato de dado. Componentes são incorporados ao programa como uma caixa preta e, como tal, não podem sofrer alterações assim Staa 2000 definiu componente.

Pressman (2002) disse que a estrutura hierárquica dos componentes de programa (módulo), o modo pelo qual esses componentes interagem e a estrutura dos dados que são usados pelos componentes definem a arquitetura de um software. Os componentes devem favorecer a facilidade de manutenção e que sejam reutilizáveis por outros softwares pois interferem em toda a arquitetura do software.

Meyer (1997) deixou claro que os componentes devem possuir características de Integridade é a habilidade de se protegerem contra acessos não autorizados.

Desta forma é possível definir para um determinado módulo os seus parâmetros de entrada e saída, os parâmetros esperados para o funcionamento de sua interface e para seu comportamento global. Sendo assim obtemos a separação da atividade de desenvolvimento de software e de sua integralização.

Com a técnica de desenvolvimento de software por componentes e técnicas de produção o desenvolvimento e testes de sistemas se tornam mais confiáveis e tolerantes a falhas.

Pressman (2002) alerta que sistemas nos quais existe forte dependência entre seus componentes podem se tornar pesadelos no caso de depuração. Os sistemas mais fáceis de alterar são considerados como aqueles constituídos por componente mais fácil de entender e o mais independente possível uns dos outros.

# 5 CONECTIVIDADE E ESTABILIDADE

A garantia de um sistema estável é a possibilidade de ser alterado um procedimento especifico sem que influenciem-se outros módulos ou procedimentos, sendo assim, são procedimentos testados e aprovados para diversas situações. Como prova disso Myers (1975) propôs um modelo de avaliação para a estabilidade de software.

Este modelo resulta em uma métrica global para o sistema que fornece a seguinte informação: ao alterar um módulo no sistema, quantos módulos serão alterados no total. O modelo tem como base teórica Probabilidade e Teoria dos Grafos.

Staa (2000) defende que a conectividade é definida como a comunicação entre módulos e deve ocorrer apenas por meio da utilização dos serviços definidos na interface do módulo. Esse princípio segue os critérios da continuidade, proteção e inteligibilidade.

Ferreira (2006) retrata que a estabilidade de um sistema é um recurso poderoso no processo de software, pois significa, dentre outros benefícios, um forte dado para a predição do esforço real necessário para a realização de alterações em sistemas. Para obter um sistema com as características de conectividade e estabilidade é necessário construí-lo a partir de módulos fortemente coesos e com o menor grau de acoplamento entre si.

Meyer (1997) disse que a melhoria da estabilidade das linguagens de programação é um fenômeno lento neste campo, mas tem sido de grande benefício para os usuários. Por esse motivo, após a década de 90 tem havido uma tentativa contínua a simplificação, clarificação e limpeza, afetando apenas detalhes, e trazendo duas recentes extensões: o mecanismo de simultaneidade e construir um precursor para facilitar a redefinição.

Pode-se concluir que a estabilidade e a conectividade em sistemas são propriedades que trabalham em conjunto e devem ser levadas em conta durante o projeto do software. Não pode deixar de avaliar a linguagem a utilizada, pois através dela se obterá os recursos necessários para um sistema de alta qualidade com a estabilidade necessária ao cliente final.

## 5.1 Acoplamento e coesão

Os conceitos de acoplamento e coesão estão diretamente ligados ao grau de conectividade, Quanto menor o grau de conectividade, mais fácil estender o módulo, menor a possibilidade de propagação de erros, mais fácil o entendimento, a manutenção e a reutilização do módulo (STAA, 2000).

De acordo com Myers (1975), a medida de grau de relacionamento entre módulos é chamada acoplamento, e a medida do grau de relacionamento entre elementos internos de um módulo é chamada coesão. O modelo ideal de acoplamento e coesão é que alteração de um componente não afeta nenhum outro módulo, caracterizando um alto grau de estabilidade.

> Os conceitos de coesão e acoplamento são utilizados na avaliação do grau de modularidade de um sistema. Bons projetos modulares apresentam alto grau de coesão e baixo grau de acoplamento, o que significa que são compostos por módulos com funções bem definidas e com pouca dependência entre si. (COSTA SANTOS, 2006, p.31).

Ferreira (2006) propõem que a coesão seja a medida do grau de homogeneidade funcional de um módulo, ou seja, a coesão indica o quão relacionados estão os procedimentos internos de um módulo.

Módulos com alta coesão têm responsabilidades bem definidas, e os com baixa coesão tratam de mais de uma responsabilidade que geralmente podem ser divididas em dois ou mais eventos. O acoplamento por sua vez mede o nível de dependência, isto é, o grau de intensidade da interconexão entre os módulos.

Ao se projetar um sistema, deve-se tentar minimizar o acoplamento para evitar a propagação de alterações decorrentes de mudanças de requisitos ou de manutenção do sistema. O baixo acoplamento também favorece a reutilização dos módulos. Quanto mais fraco o acoplamento entre os módulos, mais modular é sistema (FIG. 10).

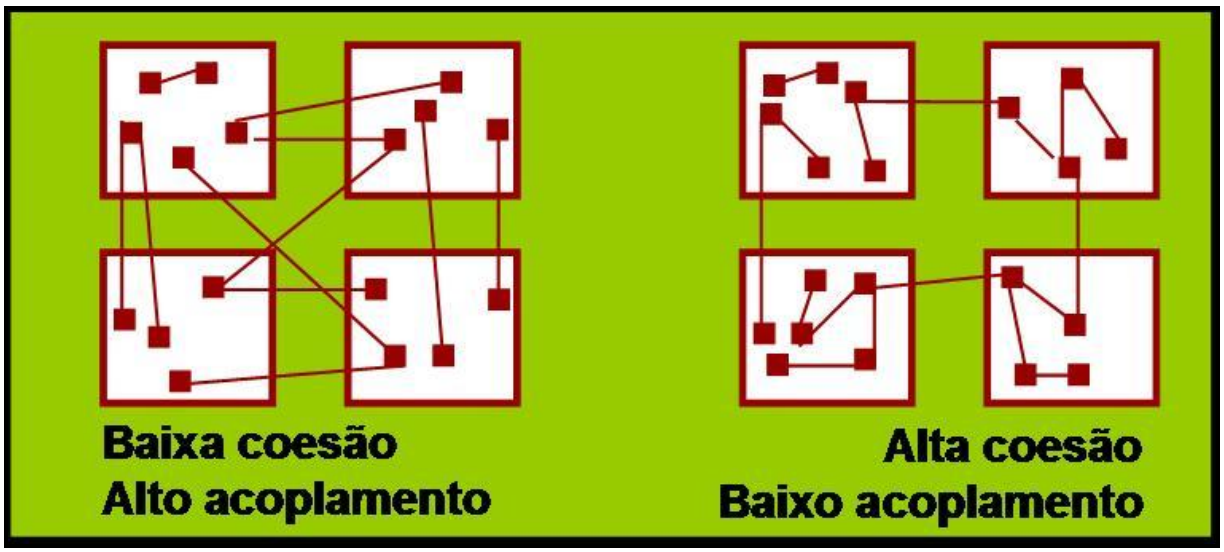


FIG.11 Modelo Coesão e acoplamento
Fonte: http://engenhariadesoftware.blogspot.com/2010/11/o-que-e-arquitetura-de-software.html

De forma geral, o acoplamento é quanto um módulo conhece do outro. Quanto mais ela conhece mais difícil será sua manutenção pois esse módulo é muito dependente do outro. Quando esse módulo for alterado o outro também pode necessitar de alteração de modo que o baixo acoplamento além de evitar efeito colateral interno evita também que um erro ou defeito se propague para os outros módulos.

Coesão é a quantidade que um módulo é específico para realizar sua tarefa. Quanto menor a diversidade de assuntos abordados por uma entidade, maior a sua coesão.

Esses conceitos tornam a manutenção mais fácil pois permite substituir um módulo por outro sem grandes transtornos e diminui a comunicação entre eles, o que gera qualidade de software com baixo acoplamento e alta coesão.

# 6 PROGRESS

O Progress é um ambiente de desenvolvimento baseado na tecnologia cliente/servidor que visa oferecer soluções informatizadas a problemas das inúmeras áreas de negócio existentes.

Embora seja uma linguagem orientada a eventos, o ambiente de desenvolvimento apresenta recursos das modernas técnicas de orientação a objetos, mesmo sendo uma linguagem orientada a eventos, visando buscar a redução de custos de desenvolvimento e manutenção, aumento de produtividade, de qualidade, eliminação de redundância de código, entre outros.

Costa (2000) define o funcionamento do Progress de forma bastante simples. A configuração típica e mais comum é a instalação e armazenamento centralizado do Progress cliente/servidor, bancos de dados e aplicações em um servidor disponível para acesso de diversos clientes em rede. A figura a seguir (FIG. 12) exemplifica uma configuração simplificada.

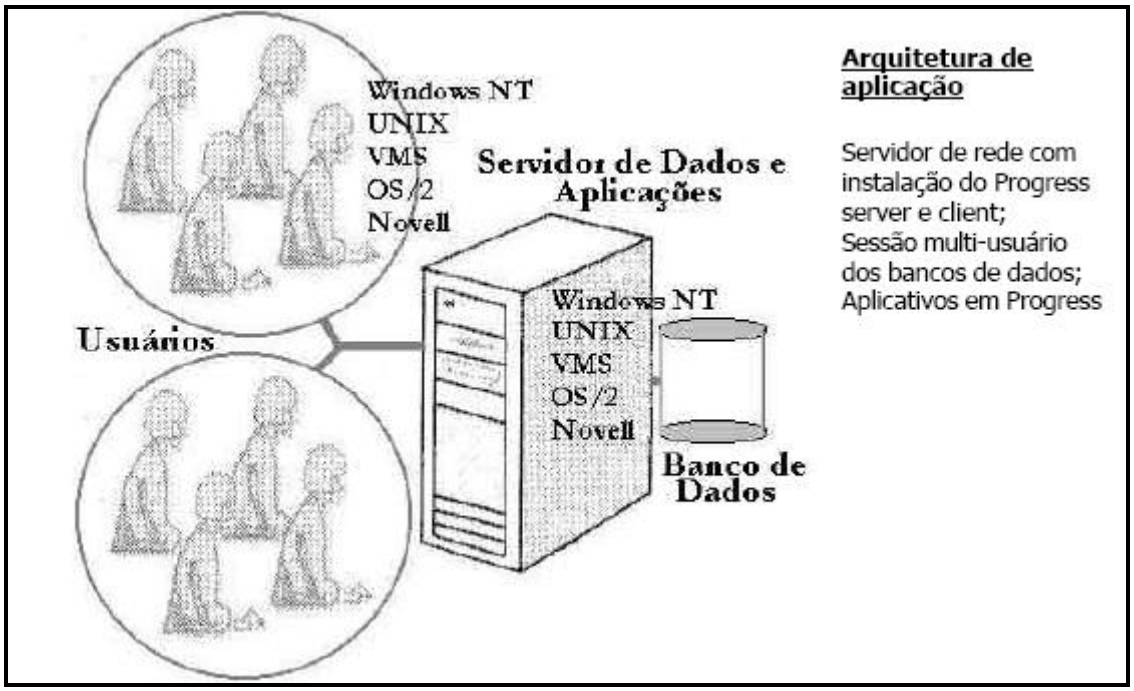


FIG. 12 Modelo de funcionamento cliente servidor do Progress
Fonte: COSTA, 2000, p.22

Conforme Costa (2000), o Progress possui ferramentas que permitem executar as tarefas de:

- Escrever código fonte;
- Executar de procedures;
- Depuração de programas;
- Compilação de qualquer tipo de código Progress;
- Administração e manutenção dos bancos de dados;
- Modelar a estrutura de tabelas, campos, índices, seqüências e triggers;
- Compilação de programas;
- Criação e edição de scripts de conexão e/ou de inicialização do Progress;
- Edição rápida e gráfica de programas baseados no Windows.

Esses atributos da linguagem permitem que todas as regras, critérios e princípios de modularidade, defendidos por Staa (2000), Meyer (1997) e outros autores sejam facilmente utilizados. Contudo é necessário que os conceitos de modularidade apresentados nesse trabalho sejam seguidos por todos os envolvidos no desenvolvimento de um sistema ou na customização de um ERP.

Abaixo temos a figura do Desktop do Progress (FIG. 13), um aplicativo desenvolvido em Progress 4GL modo gráfico que apresenta os botões dos aplicativos Progress disponíveis, Costa (2000).


FIG.13 Tela do aplicativo _Desk.p do progress
Fonte: PROGRESS Desktop Release 10.1B

Observa-se exemplo abaixo pode observar que o Progress permite utilizar o conceito de componentes, servindo de base para um reúso em larga escala no desenvolvimento de sistemas.

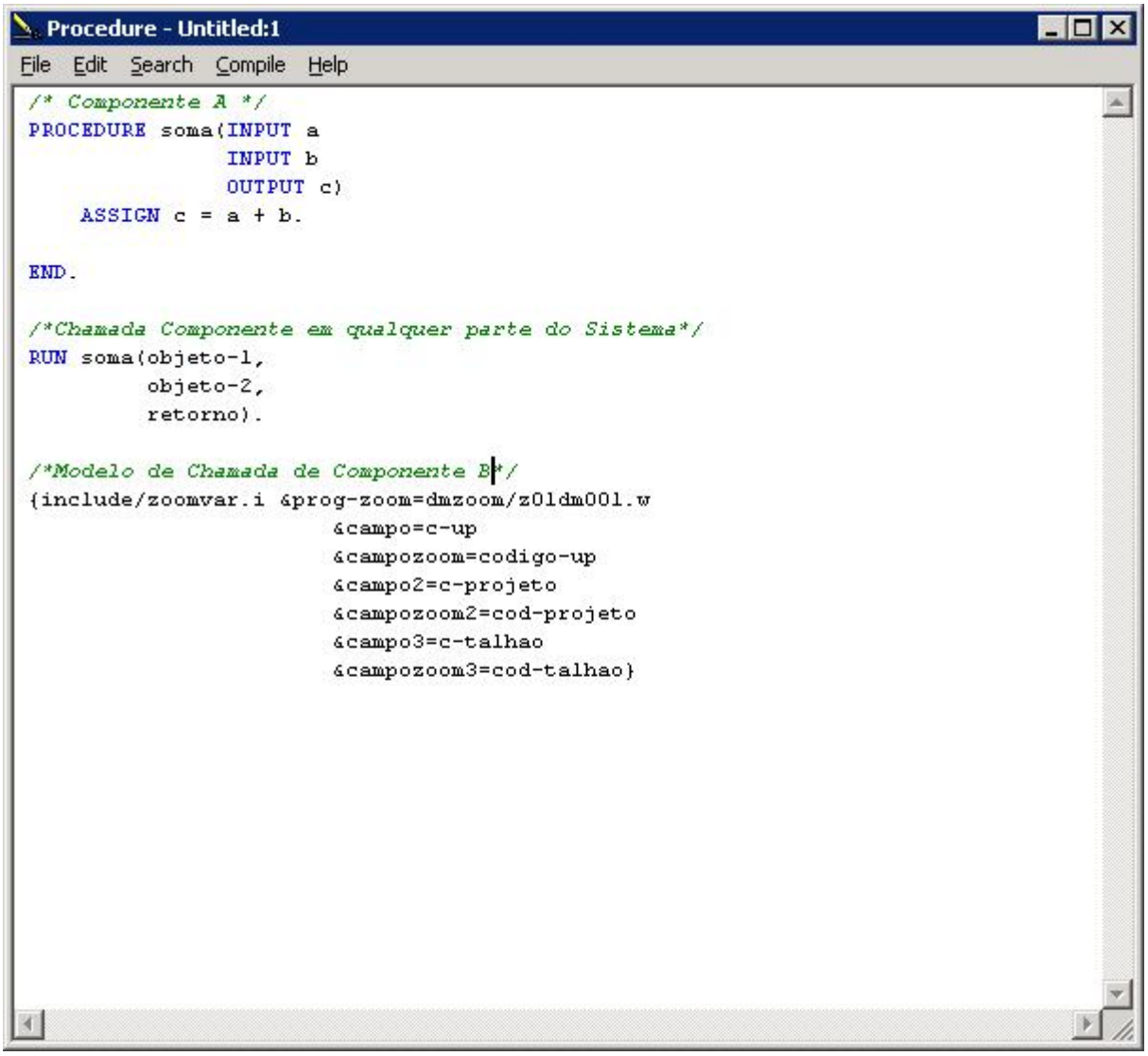


FIG.14 Exemplo de Componente Progress.

Nesse exemplo o componente A realiza a soma de dois objetos. O componente A pode ser chamado de qualquer parte do sistema e em qualquer momento, de acordo com a necessidade de qualquer outro componente. Esse componente em outras linguagens seria denominado um componente de função.

No modelo de componente B observamos um componente mais complexo, onde seria passado o nome do componente a ser chamado e os vários objetos necessários para a sua execução. Nesse caso o retorno seria apenas um ok ou nok. Com esse modelo de componente aplica-se a composibilidade pois a partir da junção de componentes distintos pode-se criar um novo sistema.

A chamada do componente exige que sejam passados os parâmetros. A passagem de parâmetros para os componentes deve sempre ser considerado os conceitos de acoplamento para garantir que o sistema atenda todos os conceitos de qualidade além das premissas de proteção que devem gerar pouca ou nenhuma propagação de erros para os demais módulos conforme Meyer(1997) citou.

De acordo com Costa (2000) o Progress permite a criação de variáveis local, semi local e global tal característica permite apoio ao controle e acoplamento, limitando aos módulos acesso a variáveis globais, conferindo-os com pouca interface entre si e com interfaces de tamanho reduzido. Conforme a NBR ISO/IEC 9126-1:2003 a confiabilidade é um requisito fundamental para a qualidade que nesse caso é atingida ao chamar o componente via parâmetro.

O conceito de conectividade descrito por Staa (2000) salienta que comunicação entre módulos deve ocorrer apenas por meio da utilização dos serviços definidos na interface do módulo. O Progress permite que esse conceito seja aplicado utilizando a passagem de parâmetros, e por ser uma linguagem clara, limpa, com alta expressividade e legibilidade limita o aumento do grau de relacionamento entre os elementos internos de um módulo, deixando a coesão em nível aceitável. Em conjunto com esse fator controla a conectividade de todo o sistema.

A estabilidade de um sistema é oferecida pelo Progress devido a linguagem que quando utilizada seguindo o modelo de componentes. Os possíveis erros são tratados dentro de seus próprios componentes passando do mais baixo para o mais alto e somente em ultimo caso para a tela principal utilizada usuário, atendendo também as premissas da qualidade externa.

Segundo Costa (2000) uma característica muito utilizada no Progress é a UPC (User Program Call) que consiste em uma interação com um programa já desenvolvido, testado e em produção. Esse programa possui um ponto de chamada onde outros programas podem chamá-lo em momento de execução transmitindo as instruções necessárias. Essa tecnologia viabiliza o reuso e a manutenibilidade do sistema, pois programas já fechados podem ser trabalhados de forma segura, via parâmetros, e ainda diminui esforço de manutenção conforme defendido por Meyer (1997).

A UPC permite também que o conceito de aberto e fechado seja bem aplicado à linguagem. Quando um módulo for fechado pode-se utilizar essas chamadas para situações especificas de utilização do módulo sem a necessidade de reabri-lo.

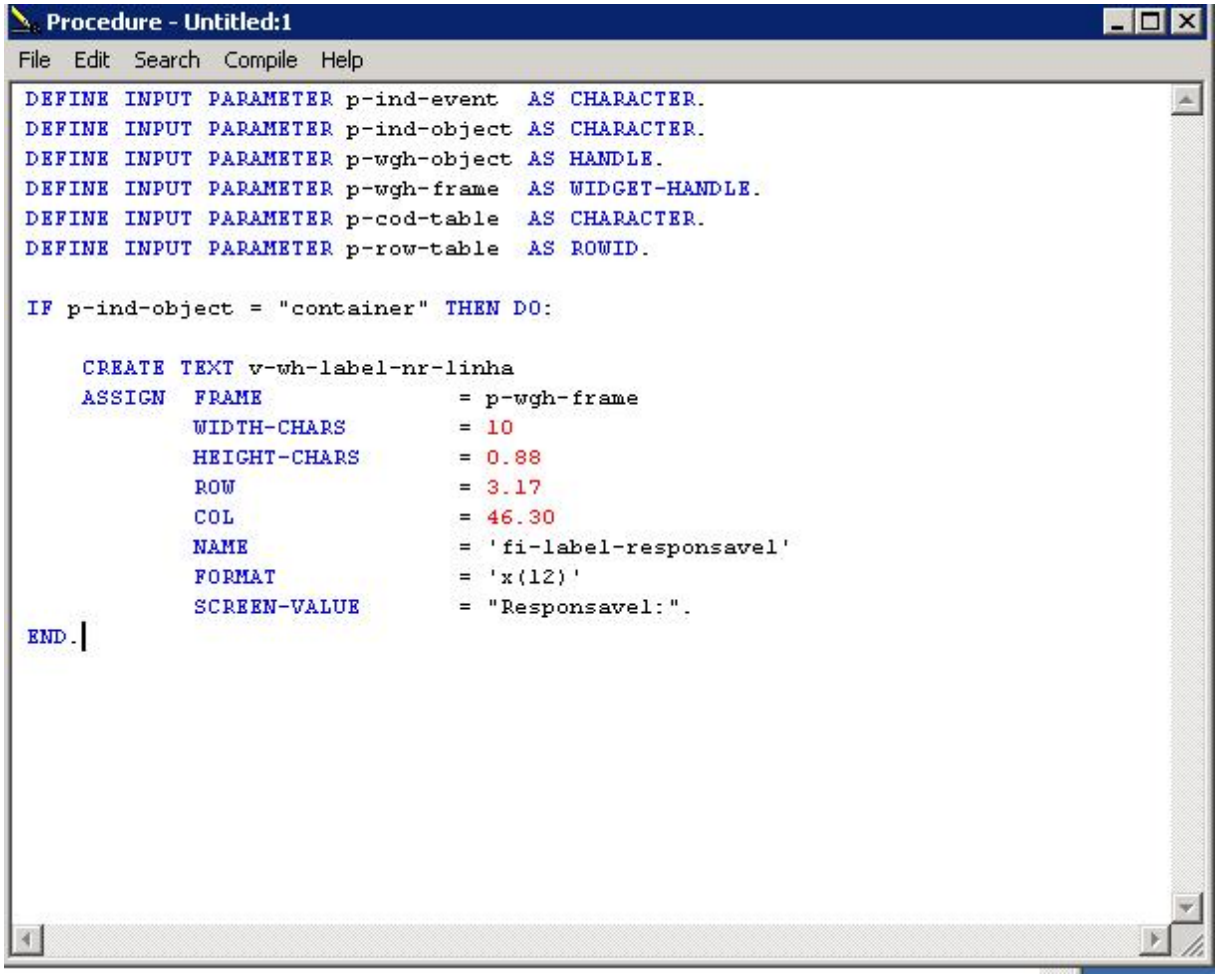


FIG.15 Exemplo de UPC.

Acima temos um exemplo de UPC, onde é possível analisar interação entre um programa já fechado e um módulo externo. O programa fechado possui esse ponto de comunicação, o programa externo realiza a comunicação através desse ponto sem atrapalhar a estabilidade do módulo já fechado traduzindo em compatibilidade entre os componentes.

O banco de dados do Progress é constituído de pelo menos dois arquivos básicos, o próprio Banco de Dados com extensão DB e o Controle de Transações com extensão BI. O arquivo de BI tem o papel de assegurar a total e completa

integridade física e referencial dos dados, como por exemplo índices, validações, tipos e formatos através do Engine Database e confirmar estas transações para gravação e/ou deleção no banco de dados após uma completa consistência dos dados existentes conforme citado por Costa (2000). Abaixo o modelo de controle do Banco de Dados (FIG. 14):

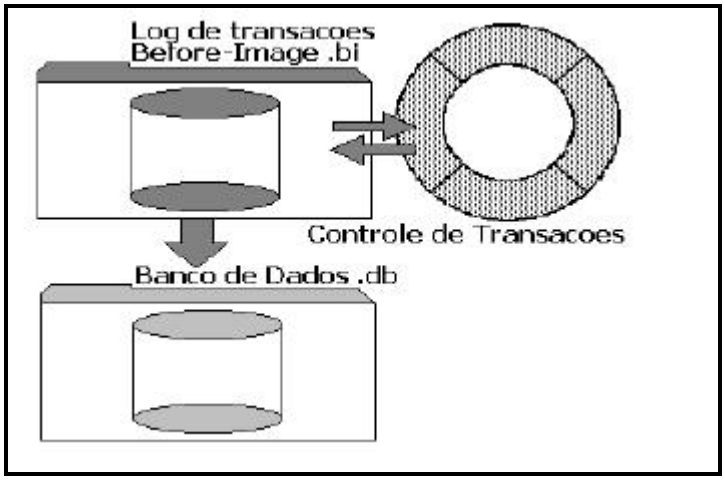


FIG.16 Modelo de controle banco de dados Progress
Fonte: COSTA, 2000, p.52

O Progress é uma linguagem versátil que possui um banco de dados nativo muito eficiente permite desenvolver sistemas ágeis, eficientes e de baixo custo, Costa (2000), por esse motivo a Totvs possui um ERP, o Data Sul, baseado nessa linguagem, pois ela permite o encapsulamento dos dados. Um módulo irá consistir de dados e procedimentos para manipular esses dados, favorecendo a eficiência e coerência na hora da manutenção do sistema.

No Progress toda a informação referente a um componente e armazenada dentro do próprio componente, como um comentário Costa (2000), assim somente os construtores daquele módulo tem acesso a essa informação. Para os componentes externos o módulo e conhecido apenas por sua interface, que seriam as informações publicas conforme defendeu Ferreira (2006).

Por possuir essas características é ter uma grande capacidade de processamento de dados o Progress trabalha com os dados de acordo com o seu tipo Costa (2000),

utiliza os conceitos de polimorfismo, por exemplo a junção de dois objetos pode ser uma soma ou uma concatenação de acordo com o seu tipo.

Como o seu banco de dados possui o BI, é possível acompanhar o grau de acoplamento e coesão do produto desenvolvido. Para que esse acompanhamento ocorra basta verificar o quanto de dados esta no BI e qual componente esta utilizando esses dados. Dessa maneira e possível controlar o acoplamento e a coesão de acordo com as boas praticas.

Costa (2000) salienta ainda que o Application Compiler possibilita compilar um diretório ou uma árvore de diretórios, especificar arquivos iniciados por uma subpalavra ou apenas uma extensão, mas sugere que apenas os componentes alterados devam ser compilados devido a demora do processo. Com isso vai de encontro com Staa (2000) que afirma quê utilizado o conceito de reuso reduz o tempo de compilação, pois somente os módulos afetados pelas alterações deverão ser recompilados.

Com a criação de componentes, utilização de UPC, chamadas via parâmetros, o uso de um banco de dados que oferece segurança além de módulos consistidos de dados e procedimentos que oferece uma compilação independente para apenas os procedimentos afetados podemos obter o mapeamento direto e uma decomposibilidade eficaz e robusta. Dessa forma é atendido os princípios de padrão de projeto tais como uma melhor comunicação entre equipes, aprendizado com a experiência, criação de arcabouços, estabilidade, evolução do código, maior produtividade entre outros citados por GAMMA et alii (2000).

Utilizando os recursos oferecidos por essa ferramenta é possível aplicar as técnicas de engenharia de software citadas criando um sistema com alto nível de qualidade, competitivo em qualidade e custo alem de obter uma continuidade garantida.

# 7 CONCLUSÃO

Sistemas de informática em geral que apoiam processos e procedimentos com alto nível de qualidade não são mais um diferencial e sim uma obrigação. Os padrões de qualidade e eficiência exigido dos sistemas atualmente já não permitem erros do tipo tela azul ou travamentos intermináveis. Para atender esses padrões de qualidade também é necessário que os softwares sejam bem projetados e com uma estrutura interna bem definida e padronizada.

O Progress é uma linguagem orientada a eventos é que possui um banco de dados nativo muito poderoso. Essa junção de fatores permite que os sistemas desenvolvidos possam usufruir de todas as técnicas de modularidade citadas nesse trabalho.

A criação de módulos ao invés de sistemas totalmente procedurais com código corrido e algo totalmente possível com o progress. Pode-se dividir todo o sistema em componentes que são chamados a partir de qualquer ponto do sistema apenas passando os parâmetros necessários. Possibilitando criar um acervo de ativos de software, tornando gerenciável o processo de desenvolvimento de versões sucessivas e cada vez mais completas.

Como o Progress possui como característica ser utilizado para o desenvolvimento de sistemas muito grandes, como ERP, a manutenibilidade do sistema e uma característica fundamental. Nesse trabalho foi apresentado como ela deve ser empregada e suas principais características que podem ser utilizadas na linguagem abordada.

Dessa maneira as técnicas de reúso, polimorfismo são passiveis de serem utilizadas. O acoplamento e a coesão são fatores que sendo analisadas por profissionais competentes podem ser controlados a favor do sistema desenvolvido.

Com o acompanhamento desses fatores é possível que qualquer profissional possa manter o sistema de forma estável e dar continuidade ao mesmo pelo tempo que for necessário e sempre evoluído atendendo assim todos os critérios de qualidade. A utilização de todos os conceitos abordados no trabalho se traduz automaticamente em um menor custo na analise e desenvolvimento dos projetos de software.

## 7.1 Contribuições da monografia

Esta monografia provê fundamentação teórica para construção sistema utilizando a linguagem progress de desenvolvimento. Como contribuição adicional, porém não menos importante, através do estudo das referências bibliográficas, buscaram-se as melhores técnicas de análise e desenvolvimento de software modular para a linguagem em questão.

## 8 REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR ISO/IEC 9126-1:* Engenharia de software - Qualidade de produto Parte 1: Modelo de qualidade. Rio de Janeiro, 2003. 21 p.

BIGONHA, Mariza A. S.; BIGONHA, Roberto S. *Programação Modular*. Belo Horizonte: Apostila, DCC-UFMG, 2001.

CERVO, A. L.; BERVIAN, P. A. *Metodologia Científica*. 5 ed. São Paulo: Prentice Hall, 2002.

COSTA, Márcio B. *Dominando o Progress*, Disponível em: < http://www.apostilando.com/download_final.php?cod=2778> Acesso em 20/07/2010.

DEITEL, H. M.; DEITEL, P. *J. Java - Como Programar.* Porto Alegre: Bookman, 2001.

FERREIRA, Kecia A. M. *Avaliação de Conectividade em Sistemas Orientados por Objetos*. Dissertação de Mestrado, DCC – UFMG, 2006.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Padrões de Projeto Soluções reutilizáveis de software orientado por objetos*. Porto Alegre: Bookman, 2000.

MEYER, Bertrand. *Object-oriented Software Construction.* Prentice Hall, 1997.

MYERS, Glenford J. *Reliable software through composite design*. Nova York: Petrocelli/Charter, 1975.

NOGUEIRA ,Carla M. *Padrões de Projetos e Princípios de Projetos*. Monografia de Final de Curso, DCC – UFMG, 2006.

PÁDUA, Wilson F. *Engenharia de Software – Fundamentos*: Métodos e Padrões. 3 Ed. Cidade: Rio de Janeiro LTC Editora, 2003.

PRESSMAN, Roger S. *Engenharia de Software*. Rio de Janeiro: MacGraw Hill, 2002.

SANTOS, Carla C. *Princípios de Programação Modular*. Monografia de Final de Curso, DCC – UFMG, 2006.

SOMMERVILLE, Ian. *Engenharia de Software*. 6 ed. São Paulo: Addison-Wesley, 2003.

STAA, A. V. *Programação modular:* Desenvolvendo problemas complexos de forma organizada e segura. Cidade: Rio de Janeiro Editora Campus, 2000.

VAREJÃO, Flávio. *Linguagens de Programação, Java, C e C++ e outras.* Rio de Janeiro*:* Editora Campus, 2004.

Yourdon, E. *Análise e projeto orientados a objetos*. São Paulo: Makron Books,1999.