UNIVERSIDADE PAULISTA

PROPOSTA DE UMA BASE DE CONHECIMENTO SOBRE COMPONENTES PARA APOIAR O DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES

ROSANGELA KRONIG

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Paulista, para obtenção do título de Mestre.

Orientador: Prof. Dr. Ivanir Costa

SÃO PAULO

2007

UNIVERSIDADE PAULISTA

PROPOSTA DE UMA BASE DE CONHECIMENTO SOBRE COMPONENTES PARA APOIAR O DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES

ROSANGELA KRONIG

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Paulista, para obtenção do título de Mestre.

Orientador: Prof. Dr. Ivanir Costa.

Área de Concentração: Gestão da Informação.

Linha de Pesquisa: Produção de Software

Projeto de Pesquisa: Ferramentas para apoiar o reuso de software

SÃO PAULO

2007

KRONIG, Rosangela

Proposta de uma base de conhecimento sobre componentes para apoiar o desenvolvimento de software baseado em componentes / Rosangela Kronig - São Paulo, 2007. 118 p.

Dissertação (Mestrado) – Apresentada ao Programa de Pós Graduação em Engenharia de Produção da Universidade Paulista, 2007.
Área de Concentração: Gestão da Informação
Linha de Pesquisa: Produção de Software
Orientador: Prof. Dr. Ivanir Costa

1. Reuso de Software. 2. Componentes de Software.
3. CBSD. 4. Biblioteca de Componentes de Software

Ao Nivaldo, pelo apoio e incentivo na busca de novos desafios À minha filha Letícia, exemplo vivo do meu amor e de Deus.

Aos meus pais, pela minha existência e que sempre me incentivaram nos estudos.

Em especial, ao Mestre Ivanir Costa, por sua atenção e dedicação.

Ao meu orientador e amigo Prof. Doutor Ivanir Costa, pelas diretrizes, orientações e incentivo para a conclusão desta dissertação.

Ao Prof. Dr. Marco Antonio Pinheiro da Silveira e Prof. Dr Marcelo Schneck de Paula Pessoa, membros da Banca Examinadora, pelas valiosas sugestões e que atenderam prontamente ao meu convite,

Ao Prof. Dr. Marcelo Schneck de Paula Pessoa, pela atenção e disposição em nortear meu ingresso no curso e durante o desenvolvimento deste trabalho.

Ao Prof. Dr. Mauro de Mesquita Spínola, pelos ensinamentos em sala de aula e considerações no desdobramento desta dissertação.

Aos professores do Programa de Mestrado em Engenharia de Produção da UNIP pelo apoio e estímulo, Prof. Dr. José Benedito Sacomano, Prof. Dr. José Paulo Alves Fusco, Prof. Dr. Antonio Roberto Pereira Leite de Albuquerque e, em especial, ao Prof. Dr. Oduvaldo Vendrametto.

Ao Prof. Dr. Antonio Carlos Gil por sua atenção e postura contributiva.

Aos alunos da graduação dos cursos de Engenharia da UNIP, participantes do projeto de pesquisa “Ferramentas para apoiar o reuso de software” do programa de Mestrado em Engenharia de Produção, Karina Nogueira de Albuquerque, Roberto Nakatsubo, e de maneira especial, aos alunos Helbert dos Santos e Janaína Santiago da Silva, pela dedicação e responsabilidade.

.Aos funcionários da secretaria da UNIP pelo atendimento prestado.

Aos colegas do mestrado, que contribuíram direta e indiretamente no enriquecimento deste trabalho, em especial Luciano Souza, Simone Canuto e Jadelson Ferreira da Silva.

Aos meus familiares, Ronaldo e Margareth Kronig, Rogério Gustavo Kronig, Alice Palmeri, Marcos Palmeri e Solange Wandeur, pelo apoio à minha vida pessoal.

Aos meus amigos Yoshimi Narita, Rita de Cássia Scarpini, Waldir Ucci, Helder de Sá, Osvaldo Morales, Ronald e Conceição Cassiolato, Vanise e Fernando Pereira da Silva, pelos estímulos na busca do aprimoramento profissional.

E aos muitos amigos aqui não mencionados, mas que foram importantíssimos para o desenvolvimento deste trabalho.

“Que você possa amar a Luz dentro de você.
E em todos que você encontrar.
E em tudo que você experimentar.”

James A. O´Brien

ÍNDICE

RESUMO

KRONIG, Rosangela. **Proposta de uma base de conhecimento sobre componentes para apoiar o desenvolvimento de software baseado em componentes** Dissertação (Mestrado em Engenharia de produção). Universidade Paulista, 2007.

Palavras-chave: **reuso de software; componentes de software; desenvolvimento de software baseado em componentes; biblioteca de componentes de software.**

A obtenção de um ambiente integrado e compartilhado tem sido uma das preocupações da engenharia de software. As organizações investem valores significativos no processo de desenvolvimento e manutenção de sistemas que atendam as necessidades atuais e futuras do negócio. A reutilização de software tem sido considerada uma forma e um desafio, para a redução do tempo e dos custos de desenvolvimento e aumento de produtividade e da qualidade do produto de software. O desenvolvimento de software baseado em componentes enfatiza a construção de sistemas a partir de componentes de software projetados e implementados, para atender as necessidades de negócio de uma forma mais rápida. Neste cenário, para o reuso efetivo de um componente é necessário identificar, registrar, padronizar e disponibilizar um conjunto de informações, que represente e documente todo o conhecimento associado a este componente. Esta base de conhecimento sobre componentes deve ser estruturada para apoiar um programa de reuso sistemático na busca e recuperação efetiva de um componente reutilizável. Este trabalho propõe um modelo conceitual de um repositório, denominado de Biblioteca de Componentes, para registrar, padronizar e centralizar esta base de conhecimento. A partir das especificações do modelo é produzido um protótipo da Biblioteca de Componentes, para demonstrar os conceitos pesquisados, no Laboratório de Pesquisa em Produção de Software - LabPPS do Programa de Mestrado em Engenharia de Produção da Universidade Paulista – UNIP.

ABSTRACT

KRONIG, Rosangela. **Proposal of knowledge base on component to support the Component Based Software Development (C.B.S.D).** Dissertation (Master of Science in Production Engineering) - Universidade Paulista, 2007.

Key-words: **software reuse; software components; component based software development; components library.**

The integrated environment earned and shared has been receiving attention in the software engineering area. The significant amount of organization investment on development and maintenance system processes should reach actual and further business demands. The software reuse has been considered a solution as well as a challenge in the reduction of time and development costs; as well as increases productivity and quality of the software product. The software development based on components emphasizes system building from software components designed and implemented to attend to business requirements fast and easily. In this scenario, the effective reuse of software components are necessary to identify, to register, to standardize and to release a set of structured information to represent and register all knowledge attached to it. This knowledge base of components must be structured to support a systematic reuse program on searches and effectively retrieve a component reusable. This proposal introduces a concept model for a repository named “Components Library”, to register, to standardize and centralize that knowledge base. Originating from the model specifications, it has been produced to make up the “Components Library” to demonstrate the concepts researched at the Software Research Laboratory of Production Engineering Masters Program at the Universidade Paulista - UNIP “Laboratório de Pesquisas em Produção de Software (LabPPS) do Programa de Mestrado em Engenharia de Produção da Universidade Paulista – UNIP”.

LISTA DE ABREVIATURAS E SIGLAS

ABNT	*Associação Brasileira deNormas Técnicas*
CASE	*Computer Aided Software Engineering*
CBD	*Component-Based Development*
CDSD	*Component-Based Software Development*
CBSE	*Component-Based Software Engineering*
COTS	*Commercial Off-The-Self*
EIA	*Electronic Industries Alliance*
ES	*Engenharia de Software*
IBM	*International Business Machine*
IEC	*International Electrotechnical Commission*
IEEE	*Institute for Electrical and Electronics Engineers*
ISO	*International Standard Organization*
OMG	*Object Management Group*
RUP	*Rational Unified Process*
SPICE	*Software Process Improvement Capability dEtermination*
SOA	*Service-Oriented Architecture*
TI	*Tecnologia da Informação*
UML	*Unified Modeling Language*

LISTA DE FIGURAS

LISTA DE QUADROS

1 INTRODUÇÃO

O desenvolvimento de software baseado em componentes, a partir do uso intenso da tecnologia orientada a objetos, está se tornando o principal modelo de desenvolvimento, com uma abordagem voltada para a produtividade e obtenção dos benefícios da reutilização.

Porém, a implementação da reutilização exige mudanças conceituais, culturais e tecnológicas no processo de desenvolvimento. Deve haver uma adequação das práticas de Engenharia de Software, em que o reuso é feito de forma *ad hoc* e aleatória, para um processo que formalize e torne a reutilização de componentes parte integrante do desenvolvimento de software.

Neste cenário, consolida-se a importância de implementar um ambiente estruturado, padronizado e um conjunto de atividades e recursos para apoiar o estabelecimento de um processo robusto de desenvolvimento baseado em componentes.

Este capítulo tem o intuito de apresentar a motivação para a elaboração deste trabalho, a justificativa do tema e os objetivos pretendidos. A estrutura da dissertação é também apresentada, no final deste capítulo com uma síntese de cada capítulo.

1.1 Natureza do Problema

Várias abordagens de desenvolvimento de software, desde a análise estruturada até a orientada a objetos, tinham como propósito reduzir o risco e o custo de desenvolvimento (VITHARANA *et al*, 2003).

O desenvolvimento baseado em componentes – *Component-Based Development* (CBD) ou a Engenharia de Software Baseada em Componentes – *Component-Based Software Engineering* (CBSE) é uma evolução dessas abordagens, pois tem como conceito construir um todo integrado a partir de partes padronizadas e independentes. Este conceito acrescenta a proposta do aumento da produtividade, na busca da redução dos tempos dos projetos de software.

Pelo fato de ser um novo paradigma da Engenharia de Software, o desenvolvimento de software baseado em componentes (*CBSD – Component-Based Software Development*) difere das abordagens tradicionais, em que todos os artefatos são construídos sem a preocupação do reaproveitamento. Todavia, o reaproveitamento é uma prática comum nas outras áreas da engenharia. Engenheiros mecânicos e elétricos não especificam um projeto para cada componentes que tenha de ser fabricado Eles implementam o projeto a partir de componentes experimentados e testados em outros sistemas (SOMMERVILLE, 2003).

Isto requer o desenvolvimento de um conjunto de componentes. Alguns são construídos intencionalmente ou adaptados, Outros são descobertos durante o desenvolvimento. Conforme KIELY (1998) *apud*

COSTA (2003), procurar um componente de software no ciclo de desenvolvimento de software é comparável ao processo de extração do ouro. Da mesma forma que o ouro vai sendo acumulado nos leitos dos rios, após ser carregado pela chuva montanha abaixo, os componentes são acumulados após muitos anos de desenvolvimento de produtos de softwares pelos desenvolvedores. O principal objetivo do desenvolvimento de software baseado em componentes é permitir que os desenvolvedores usem mais de uma vez o código escrito em qualquer linguagem, e que ele possa ser executado em qualquer plataforma.

Mas, como obter o componente adequado e por qual critério de seleção, sem perder o foco na redução de tempos e custos no desenvolvimento de um projeto de software?

1.2 Motivação do Trabalho

Atualmente, as organizações estão cada vez mais dependentes da tecnologia para a realização de seus negócios e que constitui um elemento fundamental para a obtenção de vantagens estratégicas e competitivas.

Uma das preocupações relacionadas com a produção de software é a implementação de sistemas que atendam às necessidades atuais das organizações e que sejam flexíveis para acompanhar as mudanças tecnológicas e os modelos de negócio.

De acordo com PRESSMAN (2006), a Engenharia de Software “é o estabelecimento e uso de sólidos princípios de engenharia para que se

possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais".

Este trabalho foi pensado e elaborado visando um elo de ligação entre a Engenharia de Software e a Engenharia de Produção, na medida em que estas duas áreas estão se aproximando vigorosamente, tanto nos processos produtivos quanto na tecnologia embarcada de equipamentos e máquinas. Hoje, as estratégias corporativas e as estratégias de manufatura compreendem a tecnologia como um dos instrumentos mais úteis na busca dos objetivos empresariais.

A proposta deste trabalho envolve o reaproveitamento de fragmentos de software, na figura de componentes. Um componente de software é considerado como um ativo e o reuso desse ativo padronizado, pré-construído e testado, pode agilizar o desenvolvimento e manutenção de software e garantir significativas reduções de tempo e custos.

Desta forma, o reuso de componentes justifica-se pelo fato de estar alinhado com paradigmas importantes da moderna manufatura, a manufatura responsiva e a manufatura ágil.

A manufatura responsiva, também denominada competição baseada no tempo, como o próprio nome diz, enfatiza o tempo como principal diferencial competitivo. Atingir a manufatura responsiva no processo de produção significa ser rápido, pontual e ter uma variedade de produtos (GODINHO FILHO, 2004).

A manufatura ágil aborda a capacidade da empresa em sobreviver e progredir em um ambiente de mudanças constantes. Ainda, de acordo com GODINHO FILHO (2004), o conceito de agilidade na manufatura engloba dois fatores importantes:

- responder a mudanças inesperadas e irreversíveis de forma correta e no tempo preciso;
- explorar estas mudanças como uma oportunidade e um meio de ser lucrativo.

Uma das principais estratégias chave que facilitam ou possibilitam a implementação da manufatura ágil é a gestão baseada no conhecimento. De acordo com YUSUF *et al* (1999) *apud* GODINHO FILHO (2004), a empresa voltada para o conhecimento reconhece que o conhecimento e a informação são os verdadeiros diferenciais da empresa de sucesso.

Quando as organizações inovam, elas não só processam informações com o intuito de resolver problemas existentes e se adaptar ao ambiente em transformação. Elas criam novos conhecimentos e informações quanto as soluções, e nesse processo, recriam seu meio (NONAKA & TAKEUSHI, 1997).

Buscando estes conceitos para a manufatura de software, a engenharia de software baseada em componentes propõe satisfazer as necessidades de negócio de uma forma mais rápida, ágil e econômica com

uma qualidade garantida, utilizando-se preferencialmente de partes já projetadas, testadas e implementadas.

Para apoiar a engenharia de software baseada em componentes este trabalho propõe instituir uma base de conhecimento sobre componentes reutilizáveis. Para O`DELL *et al* (2003) *apud* TURBAN *et al* (2005), conhecimento é informação em ação e ter conhecimento implica que ele pode ser exercitado para solucionar um problema que ocorreu anteriormente. E, todo conhecimento obtido dever ser compartilhado e disperso corretamente para o máximo benefício organizacional.

TURBAN *et al.* (2005), apresentam na Figura 1 o ciclo de gestão do conhecimento. Este ciclo é constituído de 6 (seis) etapas e é utilizado como parâmetro para formular a base de conhecimento sobre componentes reutilizáveis:

1. Criar conhecimento: o conhecimento é criado à medida que as pessoas determinam realizar coisas ou desenvolvem *know-how*. Às vezes, é trazido conhecimento de fora.

2. Capturar conhecimento: o novo conhecimento precisa ser identificado e representado.

3. Refinar conhecimento: o conhecimento precisa ser colocado no contexto ou classificado para que seja acionável.

4. Armazenar conhecimento: o conhecimento útil precisa ser formatado e armazenado em um repositório de conhecimento para ser compartilhado e acessado.

5. Gerenciar conhecimento: como uma biblioteca, o conhecimento precisa ser mantido atualizado. Ele precisa ser revisto para verificar se é relevante e preciso.

6. Disseminar conhecimento: o conhecimento precisa se tornar disponível em um formato padronizado a qualquer um na organização que precise dele, em qualquer lugar e a qualquer momento.

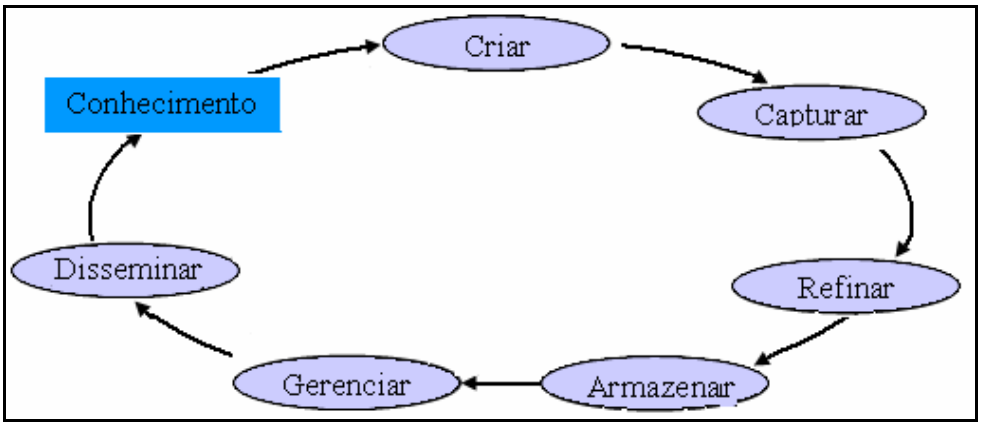


Figura 1 – Ciclo de gestão do conhecimento extraído de TURBAN *et al.* (2005, p.103)

Ainda, segundo os autores, o conhecimento nunca termina, porque o ambiente muda com o tempo e o conhecimento precisa ser atualizado para refletir as mudanças.

Para suportar um programa de reuso sistemático, vários pesquisadores apontam a necessidade de ferramentas que possibilitem o compartilhamento e acesso a componentes reutilizáveis (IEEE Std 1517-1999 (R2004); PRESSMAN, 2006; PFLEEGER, 2004; CASANOVA *et al.,* 2003).

Um repositório é uma destas ferramentas, em que o principal objetivo é tornar disponível o conhecimento sobre os componentes de software, possibilitando o compartilhamento e o gerenciamento de acesso as informações.

Para a utilização eficiente do repositório é necessário um processo para estabelecer o conjunto de informação necessário para representar e documentar um componente, implementar um esquema de classificação de informações e desenvolver mecanismos de busca e recuperação destas informações.

O presente trabalho contempla a elaboração de um modelo conceitual para representar a base de conhecimento sobre um componente de software em um repositório. O esquema de classificação de informações utilizado possibilita a aplicabilidade dos conceitos de busca e recuperação propostos por PRIETO-DIAZ (1991).

Este repositório, denominado de Biblioteca de Componentes, é implementado em um projeto piloto.

1.3 Objetivos

Os objetivos pretendidos com este trabalho são:

- Definir um conjunto de informações que identifique, classifique e documente todo o conhecimento associado a um componente de software reutilizável, estabelecendo a base de conhecimento;
- Definir um modelo conceitual que represente esta base de conhecimento sobre um componente reutilizável em um repositório, denominado de Biblioteca de Componentes, para ser compartilhado, atualizado e acessado; e
- Desenvolver um protótipo de biblioteca de componentes, baseado nas especificações do modelo proposto, como uma ferramenta para apoiar um programa de reuso sistemático na busca e recuperação efetiva de um componente reutilizável.

1.4 Metodologia do Trabalho

De acordo com OLIVEIRA (2004), uma metodologia trata do conjunto de processos pelos quais se torna possível conhecer uma determinada realidade, produzir determinado objeto ou desenvolver certos procedimentos ou comportamento. O método, derivado da metodologia, é uma forma de pensar para se chegar à natureza de um determinado problema, quer seja para estudá-lo, quer seja para explicá-lo.

O método de trabalho utilizado envolveu inicialmente o levantamento bibliográfico e análise de outros trabalhos relacionados com o tema para proporcionar uma maior familiaridade com o problema a fim de torná-lo mais explícito.

O levantamento bibliográfico foi desenvolvido com base em material já elaborado, constituído principalmente de livros e artigos científicos. Estes últimos representam nos tempos atuais uma das mais importantes fontes bibliográficas (GIL, 2002).

A partir da obtenção dos conhecimentos técnicos foi elaborado um modelo conceitual para estruturar a base de conhecimento sobre componentes em um repositório, denominado de Biblioteca de Componentes.

A abordagem por modelos é a mesma utilizada por arquitetos na especificação e projeto de edificações que constroem modelos em escala, para que se possa visualizar o sistema futuro. Estes modelos servem como veículos de comunicação e negociação entre usuários, desenvolvedores, patrocinadores, entre outros. (YOURDON & ARGILA, 1999).

SLACK *et al.* (1997) definem projeto, na Administração da Produção, como um processo conceitual para atender algumas exigências funcionais de pessoas, individualmente ou em massa, através do uso de um produto ou de um sistema que deriva na tradução física do conceito. O projeto começa com um conceito e termina na tradução desse conceito em uma especificação de algo que pode ser produzido.

A tradução física dos conceitos apresentados no modelo conceitual terminou com a especificação de um protótipo desenvolvido no Laboratório de Pesquisa em Software do Programa de Mestrado em Engenharia de Produção da Universidade Paulista – UNIP. Um protótipo é uma versão inicial de um sistema de software e normalmente leva a melhorias na especificação do sistema (SOMMERVILLE, 2003).

O Laboratório de Pesquisa, através dos trabalhos desenvolvidos no programa de mestrado, tem como objetivo estabelecer um local de pesquisa em métodos, processos e ferramentas de desenvolvimento de software,

1.5 Estrutura do Trabalho

Este trabalho foi desenvolvido com a seguinte estrutura:

O **capítulo 1** (um) – A partir de uma introdução sobre os principais aspectos envolvidos com desenvolvimento de software baseado em componentes, são descritos neste capítulo o objetivo e a justificativa deste trabalho.

O **capítulo 2** (dois) apresenta uma revisão conceitual para o estabelecimento das definições e conceitos envolvidos com o reuso de software na Engenharia de Software.

O **capítulo 3** (três) fornece o embasamento teórico para o entendimento dos aspectos envolvidos com o reuso de componentes de software.

O **capítulo 4** (quatro) aborda o ambiente de desenvolvimento de software baseado em componentes.

O **capítulo 5** (cinco) mostra a proposta do modelo de biblioteca para apoiar o reuso de componentes de software.

O **capítulo 6** (seis) apresenta a conclusão e discute as contribuições relevantes para outras pesquisas.

Finalmente, o **capítulo 7** (sete) apresenta a lista das referências bibliográficas que foram pesquisadas e referenciadas durante os estudos e montagem deste trabalho.

A Figura 2 apresenta a estrutura deste trabalho.

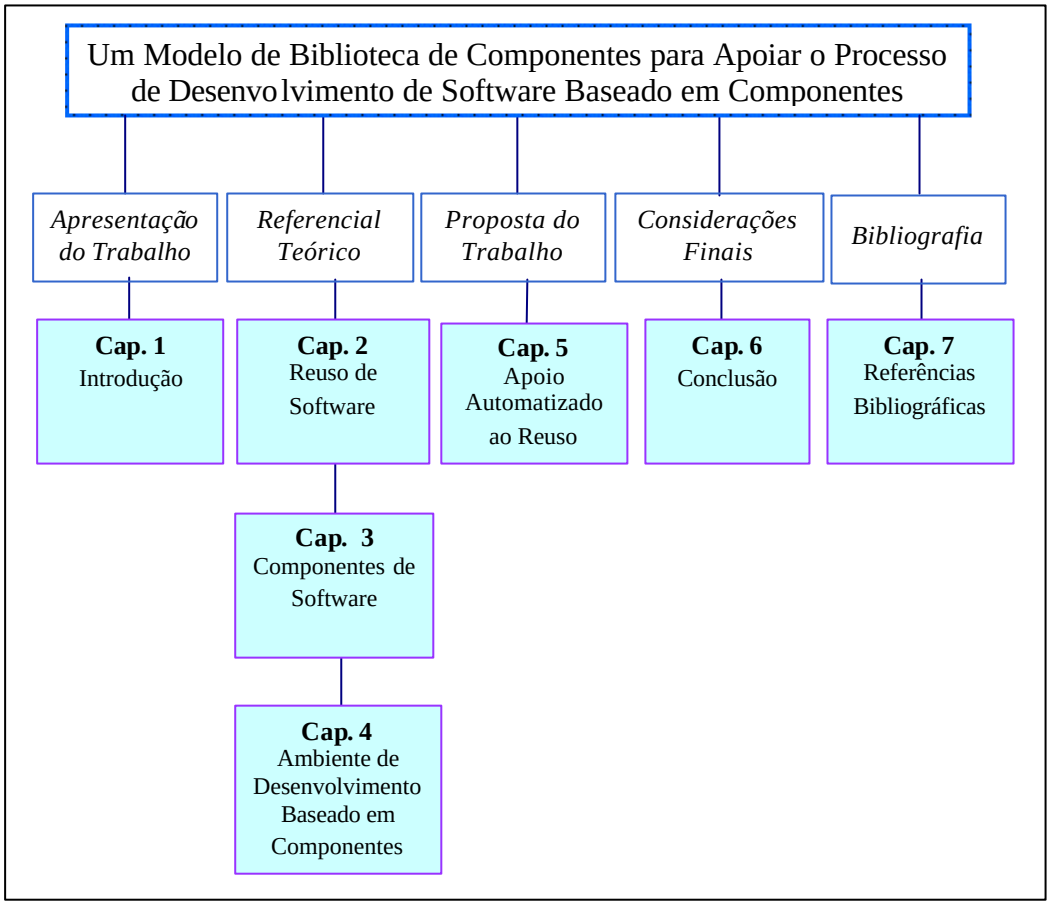


Figura 2 – Estrutura do Trabalho

2 REUSO DE SOFTWARE

A solução de um problema, muitas vezes, pode estar baseada em uma solução similar que já foi aplicada ou adaptada para resolver um outro problema. O mesmo conceito pode ser aplicado no desenvolvimento de software.

Muitos dos sistemas de software desenvolvidos possuem características comuns e propósitos semelhantes. Portanto, estes sistemas devem ser considerados na construção de um novo sistema, avaliando-se seus componentes e determinando suas adaptações ou até mesmo reutilizando-os por completo (PFLEEGER, 2004).

Para SOMMERVILLE (2003) o reuso de sistemas de software tem sido praticado há muitos anos à medida que estes sistemas são implementados em uma série de máquinas e adaptados para diferentes ambientes. Para o autor, na maioria dos projetos de software ocorre algum reuso de software. Em geral, acontece informalmente, quando as pessoas que trabalham no projeto conhecem, por exemplo, códigos similares àquele exigido. Estes códigos são modificados conforme necessários e incorporados em seus novos sistemas.

O reuso é visto como uma abordagem evolucionária essencial da engenharia de software, pois reduz a duplicação de esforços e tempos pela construção de um trabalho anterior e possibilita o aperfeiçoamento dos

produtos desenvolvidos pela sua integridade e confiabilidade (SELBY, 2005; SOMMERVILLE, 2003; McCLURE, 2001).

Para a implementação de uma abordagem de reuso na produção de software é necessário sistematizar um processo para reconhecer, classificar e registrar um conjunto de informações como uma base de conhecimento sobre os ativos desenvolvidos a ser compartilhada e disponibilizada para projetos futuros (IEEE-Std 1517–1999(R2004); SOMMERVILLE, 2003; CHEESMAN & DANIELS, 2001).

Este capítulo apresenta inicialmente os conceitos e definições sobre o reuso de software, a engenharia de software e os principais processos de desenvolvimento de software relacionados com reuso e a importância da criação e manutenção do conhecimento sobre ativos reutilizáveis.

2.1 Conceitos e definições

Entende-se por reuso o uso de um artefato na solução de problemas diferentes (IEEE-Std 1517–1999(R2004)) e por reuso de software, um ativo de software desenvolvido em outro lugar - em outro projeto ou em outra organização e utilizado em mais de um contexto, com ou sem modificações (MORISIO *et al.*, 2000).

McCLURE (2001) define reuso de software como o processo de construção ou montagem de sistemas de software a partir de partes de software projetadas para reuso. O reuso de software é praticado para ganhar tempo e dinheiro, e para melhorar a qualidade.

SAMETINGER (1997) apresenta a proposta de reuso de software como o processo de criação de sistemas de software a partir de software existente ao invés de construí-los do nada.

Para FRAKES & KANG (2005), o reuso de software é o uso de ativos reutilizáveis, que pode ser um software existente ou o conhecimento, para construir um novo software.

PRIETO-DÍAZ (1993) define reuso de software como “o uso de um componente de software existente para construir novos sistemas”.

Um outro conceito importante é o de reusabilidade que estabelece o grau para o qual um ativo de software pode ser utilizado em mais de um sistema de software ou na construção de outros ativos de software (IEEE-Std 1517–1999(R2004)).

O reuso de software tem sido praticado desde que a programação foi inventada, utilizando-se de códigos-fonte, sub-rotinas e algoritmos para criar novos programas (PRIETO-DÍAZ, 1993; FRAKES & KANG 2005).

Segundo um histórico apresentado por PRIETO-DÍAZ (1993), o conceito formal de reuso de software foi introduzido por Dough McIlroy em 1969, que propôs uma indústria de componentes de prateleira (hoje mais conhecidos como *Commercial Off-The-Shelf (*COTS)), componentes de código-fonte padronizados, vislumbrando a construção de sistemas complexos a partir da utilização de pequenos blocos construídos e disponíveis através de catálogos.

Ainda de acordo com PRIETO-DÍAZ (1993), a idéia sobre o reuso de software apresentada por McIlroy foi fundamental para o conceito de fábrica de software. Em 1974, a *System Development Corporation - SDC* registrou o termo “fábrica de software”, em que a proposta desta empresa estabelecia um conjunto de procedimentos bem definidos para um processo de software repetível e consistente, porém a reutilização de software era implícita (CUSUMANO,1991 *apud* PRIETO-DIAZ, 1993).

A experiência da SDC foi usada como base para as futuras fábricas de software, especialmente no Japão, onde processos de reuso foram mais tarde definidos e integrados.

Na década de 1970, Robert Lanergan iniciou um projeto de reuso na *Raytheon Company*, o qual foi considerado o primeiro caso de reuso em uma organização. A empresa observou que 60% das aplicações de negócio em COBOL eram redundantes e, conseqüentemente, eram indicadas para padronização e reutilização. Após a implantação do projeto, a *Raytheon* relatou que um sistema novo continha 60% de código reutilizado, aumentando a produtivdade em 50%. Nesta época, a reutilização de software era focada no código-fonte. (LANERGAN & GRASSO,1984 *apud* PRIETO-DIAZ, 1993; PFLEEGER, 2004).

Durante a década de 1980, foram criados vários programas para incentivo do reuso de software. Empresas japonesas como a NEC, a Hitachi, a Fujitisu e a Toshiba, tinham estabelecido ambientes de desenvolvimento de software integrados, que apoiavam o reuso. Empresas comerciais, como

Hewlett-Packard, Motorola e IBM também investiram na reutilização. (PRIETO-DIAZ, 1993; PFLEEGER, 2004).

Ao final deste período as pesquisas realizadas tiveram maior intensidade nas áreas relacionadas aos sistemas de bibliotecas, às técnicas de classificação, criação e distribuição de componentes , aos ambientes de suporte ao reuso e aos programas de reuso corporativos (PRIETO-DIAZ, 1993).

Diversos avanços na Engenharia de Software possibilitaram o reuso mais viável. As técnicas de projeto e implementação orientadas a objeto estimularam a construção de sistemas, em que os conceitos de herança e especialização poderiam ser aplicados para o reuso de classes e objetos (PRESSMAN, 2006; FRAKES & KANG, 2005; CHEESMAN & DANIELS, 2001).

Segundo SCHACH (2002) *apud* COSTA (2003), “o paradigma da orientação a objetos traz enormes benefícios ao desenvolvimento de software, pois ela torna o desenvolvimento e a manutenção muito mais fáceis. A aplicação correta do paradigma de objetos, com uso de encapsulamento de componentes, dos mecanismos de herança e polimorfismo tem produzido softwares cada vez mais simples, possibilitando a aplicação intensiva do reuso”.

2.2 Engenharia de Software

O IEEE *Computer Society* define Engenharia de Software como "(1) a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software, isto é, a aplicação da engenharia ao software. (2) O estudo de abordagens como as de (1)" (IEEE Std 610.12-1990).

De acordo com SOMMERVILLE (2003), a engenharia de software é uma disciplina da engenharia responsável por todos os aspectos da produção de software, desde a sua especificação até a sua manutenção. Nesta definição, segundo o autor, há duas frases importantes:

1. "Disciplina da engenharia": os engenheiros fazem os produtos funcionarem aplicando as teorias, métodos e ferramentas apropriadas, procuram descobrir soluções para os problemas e reconhecem que precisam trabalhar de acordo com as restrições organizacionais e financeiras.

2. "Todos os aspectos da produção de software": além das atividades técnicas, a engenharia de software também se dedica às atividades de gerenciamento de projetos e ao desenvolvimento de ferramentas, métodos e teorias que dêem apoio à produção de software.

Para PRESSMAN (2006), a engenharia de software é uma tecnologia em camadas. De acordo com a Figura 3, qualquer abordagem de engenharia (inclusive a engenharia de software) deve ser apoiada em um

compromisso organizacional com a qualidade. A base em que se apóia é o foco na qualidade.

O fundamento da engenharia de software é a camada de processos. Os processos formam a base para o controle gerencial dos projetos e estabelecem o contexto nos quais os métodos técnicos são aplicados, os produtos de trabalho são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente administradas.

Os métodos fornecem a técnica de "como fazer" para construir softwares. Os métodos incluem um conjunto de tarefas que abrangem comunicação, análise de requisitos, modelagem de projetos, construção de programas, testes e manutenção.

E finalmente, as ferramentas fornecem apoio automatizado ou semi-automatizado para o processo e os métodos.

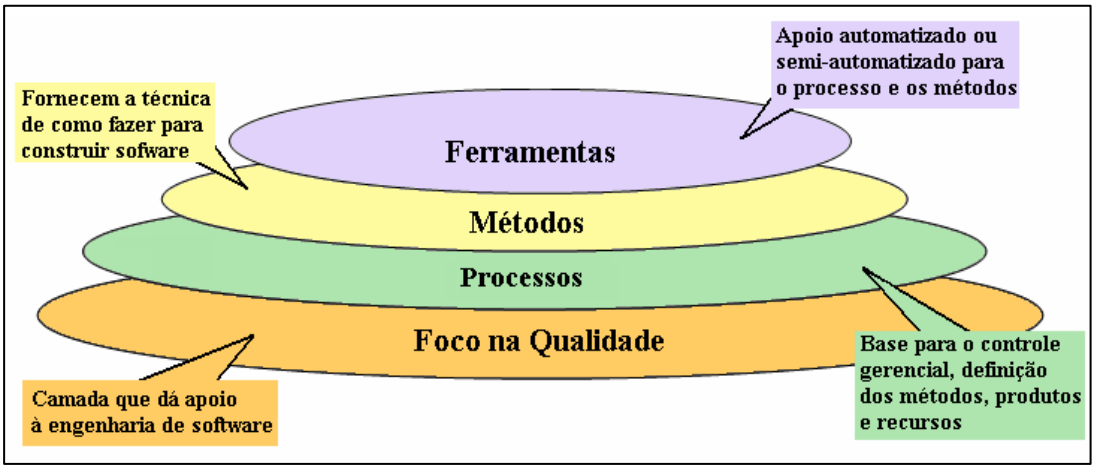


Figura 3 – Engenharia de Software em camadas adaptado de PRESSMAN (2006, p.17) .

À medida que uma disciplina de engenharia evolui, uma coleção de métodos, modelos, técnicas e ferramentas são criadas. No ambiente evolutivo da Engenharia de Software, ROYCE (1998) apresenta as três gerações do desenvolvimento de software e suas principais características:

1. Convencional – Entre as décadas de 1960 e 1970, focado no conhecimento humano. As organizações utilizavam-se de ferramentas e processos desenvolvidos e utilizados internamente e, virtualmente os componentes eram construídos em linguagens de máquinas. A ênfase era na *performance* do software e os prazos, custos e a qualidade do projeto nem sempre eram observados.

2. Transição: Entre as décadas de 1980 e 1990, focada na engenharia de software. As organizações utilizavam-se de processos mais repetitivos e ferramentas de mercado (*off-the-shelf*) e a maioria dos componentes (mais de 70%) eram desenvolvidos em linguagens de alto nível. Alguns componentes (menos de 30%) estavam disponíveis como produtos comerciais, incluindo sistemas operacionais, sistemas gerenciadores de banco de dados, redes e interfaces gráficas. Durante a década de 1980, algumas organizações passaram a alcançar uma economia de escala, mas com o crescimento de aplicações complexas (principalmente com surgimento dos sistemas distribuídos), as

linguagens, as técnicas e a tecnologia ainda não eram suficientes para garantir o desempenho desejável.

3. Práticas modernas: De 2000 em diante, focado na produção de software, com o uso de processos gerenciados e mensuráveis, ambiente automatizado e integrado, e principalmente (70%) de componentes comerciais prontos para uso ("componentes de prateleira – COTS"). Com os avanços na tecnologia de software e ambientes de produção integrados, os sistemas baseados em componentes são produzidos mais rapidamente.

O Quadro 1 apresenta as características de cada geração, de acordo com os parâmetros ROYCE (1998):

Quadro 1 - Características das gerações de software adaptado de ROYCE (1998, p.23)

	Convencional 1960 - 1980	**Transição 1980 - 1990**	**Práticas modernas 2000 -**
Ambientes/ ferramentas	Customizados	*Off-the-shelf*, não integrado	*Off-the-shelf,* integrado
Tamanho	100% customizados	30% baseado em componentes 70% customizados	70% baseado em componentes 30% customizados
Processo	*Ad-hoc*	Repetitivo	Gerenciado e medido

- **Ambientes**: composto de ferramentas e técnicas disponíveis para suportar eficientemente o desenvolvimento de software e para automatizar o processo;

- **Tamanho**: quantifica o número de instruções fontes ou o número de pontos de funções exigido para atender as funcionalidades do sistema;
- **Processo**: como o desenvolvimento do produto final é conduzido.

Segundo McCLURE (2001), padrões são essenciais não apenas para os avanços das práticas da engenharia de software, mas também para o crescimento da indústria de componentes. Portanto, os padrões devem ser integrados por meio de um conjunto de terminologia comum da engenharia de software e por uma visão comum do ciclo de vida de software.

O IEEE *Computer Society* define ciclo de vida de software como o período de tempo que começa quando o produto de software é concebido e termina quando o software não está mais disponível para uso. O ciclo de vida de software tipicamente inclui as fases de concepção, requisitos, *design*, implementação, teste, instalação e monitoramento, operação e manutenção e a fase de encerramento. Estas fases podem se sobrepor ou podem ser executadas iterativamente (IEEE Std 610.12-1990).

Processos de ciclo de vida de software são estabelecidos como forma de melhorar a qualidade do produto, facilitar o entendimento e a comunicação humana, suportar o gerenciamento e a melhoria do processo, prover um guia para execução e automação do processo (SWEBOK, 2004).

2.3 Processos de Ciclo de Vida de Software

Segundo o IEEE *Computer Society,* processo é uma seqüência de passos executados para um determinado propósito (IEEE Std 610.12-1990).

Um processo deve ser definido para a situação na qual ele será utilizado, considerando-se variáveis importantes que incluem a natureza do trabalho (por exemplo, desenvolvimento ou manutenção), infra-estrutura (recursos humanos e tecnológicos), modelos de ciclo de vida e maturidade da organização.

Um modelo de ciclo de vida é uma estrutura contendo atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término do seu uso (IEEE/EIA 12207.0-1996).

Os modelos de ciclo de vida de software definem genericamente as fases que ocorrem durante o desenvolvimento ou manutenção de software. Estes modelos não apresentam definições detalhadas, mas apresentam as principais atividades e suas interdependências (SWEBOK, 2004).

Segundo SAMETINGER (1997), os modelos de ciclo de vida foram desenvolvidos para orientar as equipes na criação de software de alta qualidade a custos previsíveis. Estes modelos foram adaptados, e várias mudanças e melhorias foram sugeridas desde o clássico modelo cascata (*waterfall model*), passando pelo evolucionário, incremental até o modelo de

ciclo de vida em espiral, proposto por BOHEM (1988) e adotado por grandes empresas de software.

A princípio, um processo de ciclo de vida de software pode ser organizado para utilizar qualquer modelo de ciclo de vida de software (SWEBOK, 2004).

Em 1995, o *International Organization for Standardization* (ISO) e o *International Electrotechnical Commission* (IEC) publicaram a ISO/IEC 12207 – *Standard for Information Technology - Software Llife Cycle Processes* para estabelecer uma estrutura (*framework*) comum para os processos de ciclo de vida de software, com terminologia bem definida, para ser referenciada pela indústria de software (IEEE/EIA-12207.0-1996) .

A estrutura contém um conjunto de processos, atividades e tarefas para a aquisição de um sistema que contém *software*, de um produto de software independente ou de um serviço de software, e para o fornecimento, desenvolvimento, operação e manutenção de produtos de software.

Conforme McCLURE (2001), os principais objetivos da norma ISO/IEC 12207 são:

- Estabelecer uma estrutura comum para processos de ciclo de vida de software;
- Melhorar a comunicação entre as partes envolvidas no ciclo de vida de um produto de software por meio de uma terminologia comum;

- Definir os processos que são aplicados durante o ciclo de vida completo de um produto de software, desde a sua aquisição até o término de seu uso;
- Capacitar o comércio mundial de produtos de software.

A partir da ISO/IEC 12207, O IEEE (*Institute of Electrical and Electronics Engineers* e o EIA (*Electronic Industries Alliance*) publicaram a norma IEEE/EIA-12207.0-1996. Esta norma consiste de esclarecimentos, adições e mudanças aceitas pelo IEEE e o EIA e de conceitos e diretrizes para promover o seu entendimento e sua aplicação. Uma das razões para sua criação foi desenvolver uma versão para os Estados Unidos, incorporando as melhores práticas e experiências da engenharia de software americana.

Em 1997, a ABNT (Associação Brasileira de Normas Técnicas) publicou uma versão traduzida para o Brasil, a norma NBR ISO/IEC 12207/1997. O objetivo desta norma, assim como a ISO/IEC 12207, é prover uma estrutura comum, e que possa ser usada por profissionais de *software* na criação e gerência de software. A estrutura cobre o ciclo de vida de *software* desde a concepção de idéias até a descontinuação do mesmo, e consiste dos processos de aquisição e fornecimento de produtos e serviços de *software*. Adicionalmente, a estrutura provê o controle e a melhoria destes processos.

Em 1998, é publicada a norma ISO/IEC TR 15504, *Information Technology – Software Process Assessment,* também conhecida como SPICE (*Software Process Improvement Capability dEtermination*) para avaliação de processo de software (KOHAN, 2003).

Em 1999, para atender especificamente ao reuso de software é definida pelo IEEE *Software Engeneering Standards Committee* uma estrutura comum para integrar a prática do reuso sistemático aos processos de ciclo de vida de software tradicionais, através da norma IEEE Std 1517-1999 - *IEEE Standard for Information Technology – Software Life Cycle Process – Reuse Process* . Esta norma foi desenvolvida para ser utilizada e integrada a norma IEEE/IEA 12207.0-1996.

A norma IEEE Std 1517-1999 (2004) define reuso sistemático, como a "prática do reuso de acordo com um processo bem definido e repetível".

Em 2001, é elaborada a emenda de norma ISO/IEC 12207:1995/PDAM 1 - *Information Technology – Software Life Cycle Processes - Amendment* 1, para ser incorporada à ISO/IEC 12207 diversos processos, entre eles, da norma ISO/IEC TR 15504 para avaliação de processo de software e da IEEE Std 1517-1999(2004) para atender a prática do reuso sistemático.

Em 2002, é publicada a ISO/IEC 12207:2002/FDAM 2 - *Information Technology – Software Life Cycle Processes - Amendment* 2 para ajustes na emenda anterior.

A Figura 4 foi elaborada para um melhor entendimento da origem e integração das normas que definem os processos de ciclo de vida de software.

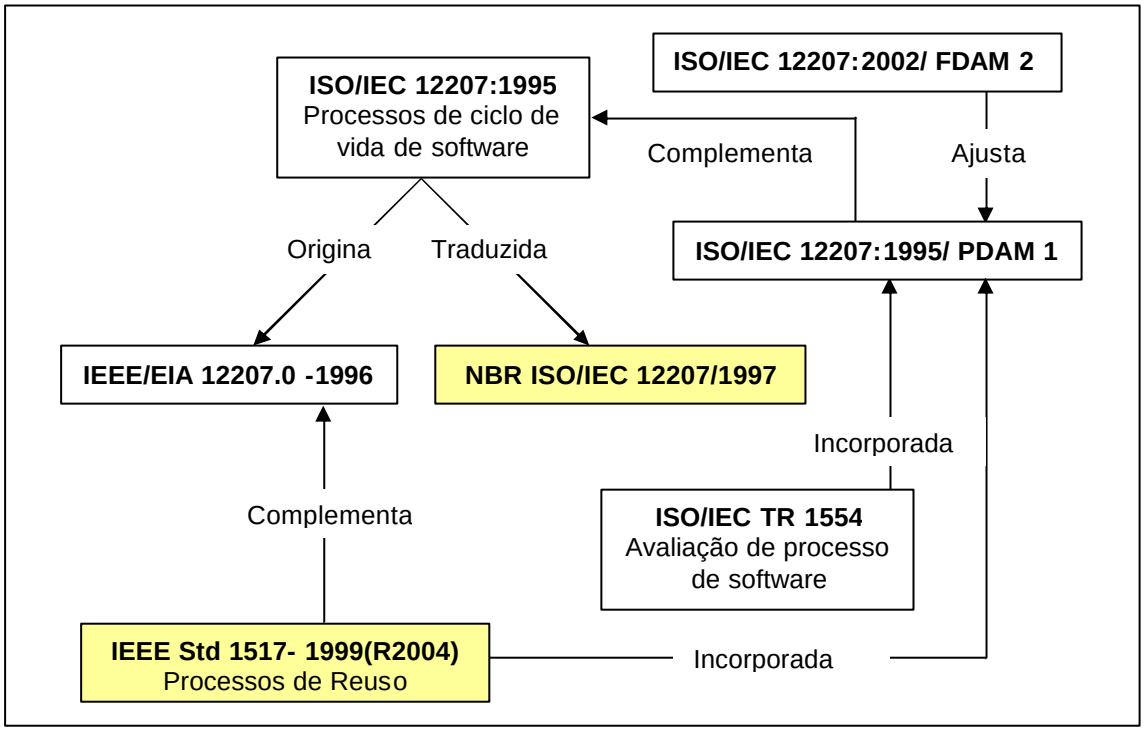


Figura 4 – Normas referentes aos processos de ciclo de vida de software elaborada pela autora

No momento, a ISO está elaborando uma nova norma para os processos de ciclo de vida de software.

Neste trabalho, é apresentado uma visão geral da norma NBR ISO/IEC 12207 e com maior detalhe a estrutura do ciclo de vida de software da norma IEEE Std 1517-1999(2004), para a prática do reuso.

2.3.1 Processos da norma NBR ISO/IEC 12207/1997

A estrutura da norma NBR ISO/IEC 12207/1997 contém um conjunto de processos, atividades e tarefas para aquisição, fornecimento, desenvolvimento, operação e manutenção de produtos de software.

Cada processo é dividido em um conjunto de atividades e cada atividade em um conjunto de tarefas. Esta organização pode ser verificada na Figura 5.

Os processos de Ciclo de Vida de software são organizados em três grandes grupos:

- **Processos Fundamentais de Ciclo de Vida** – Conjunto de atividades técnicas que englobam a aquisição, o desenvolvimento, a operação de um software até a manutenção do mesmo.
- **Processos de Apoio de Ciclo de vida** – Conjunto de atividades de apoio aos processos fundamentais para garantir que o software produzido esteja em conformidade com os requisitos especificados e aderentes aos planos estabelecidos.
- **Processos Organizacionais de Ciclo de vida** – Conjunto de atividades que estabelece e implementa a infra-estrutura dos processos fundamentais, incluindo a gerência de projetos.

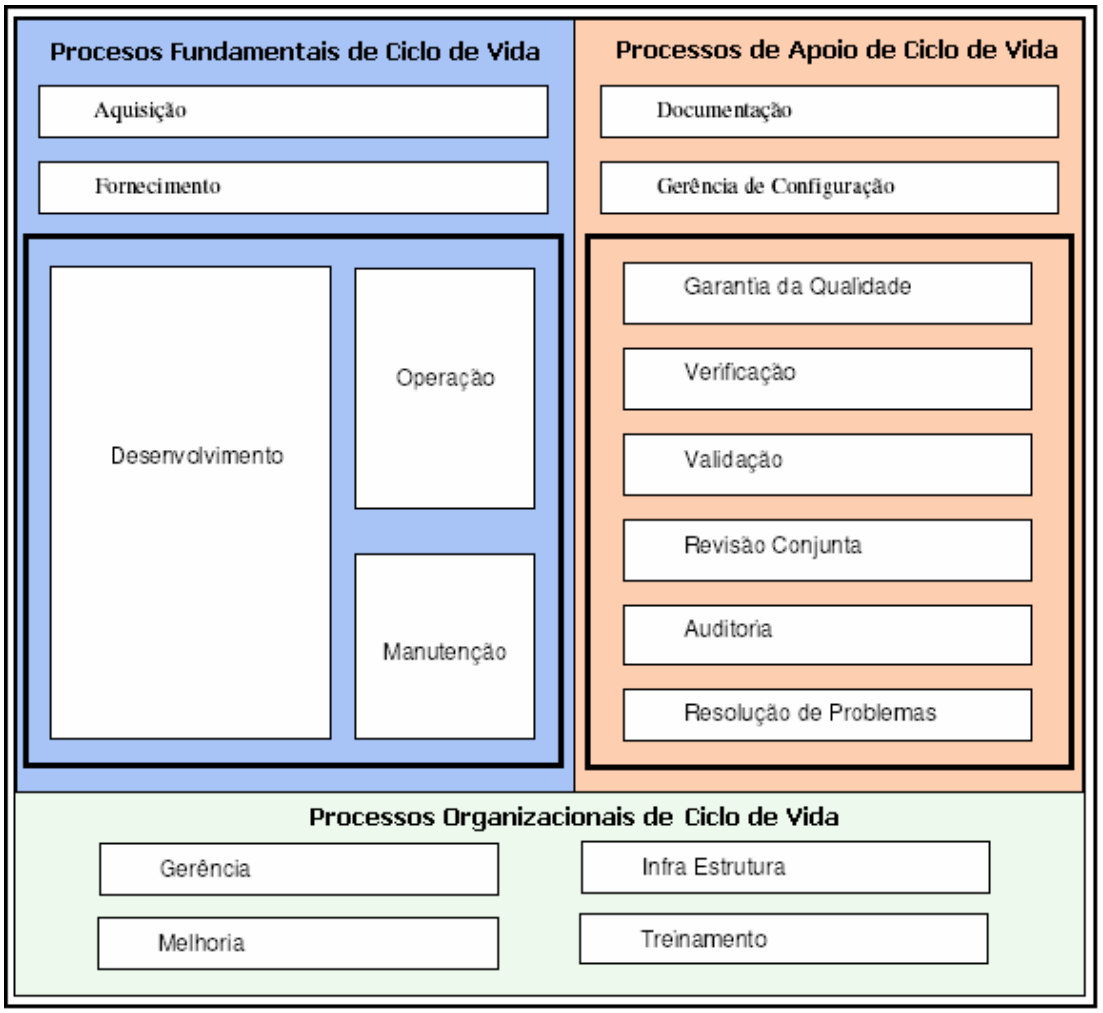


Figura 5 – Estrutura da Norma NBR ISO/IEC 12207/1997

Na norma NBR ISO/IEC 12207/1997, os processos são descritos em alto nível para permitir que a norma possa ser adaptada para qualquer aplicabilidade**.** Esta norma não é específica para um modelo de ciclo de vida de software ou uma metodologia.

2.3.2 Processos da norma IEEE Std 1517-1999(R2004)

A Implementação do reuso de software requer a formalização da prática do reuso, com a integração dos processos e atividades no ciclo de vida de software e, com a criação e uso de bibliotecas de ativos reutilizáveis.

A norma IEEE Std 1517-1999(R2004) - *IEEE Standard for Information Technology – Software Life Cycle Process – Reuse Process* define uma estrutura comum para integrar a prática do reuso sistemático nos processos de ciclo de vida de software tradicionais.

Esta norma especifica os processos, atividades e tarefas a serem aplicados durante cada fase do ciclo de vida de software para a construção de um produto de software a partir de ativos. Também especifica os processos, atividades, e tarefas para identificar, construir, manter e administrar os ativos desenvolvidos.

A Figura 6 apresenta a estrutura do ciclo de vida de software para a prática do reuso, com os processos, atividades e tarefas integrados a estrutura da norma IEEE/EIA-12207.0-1996.

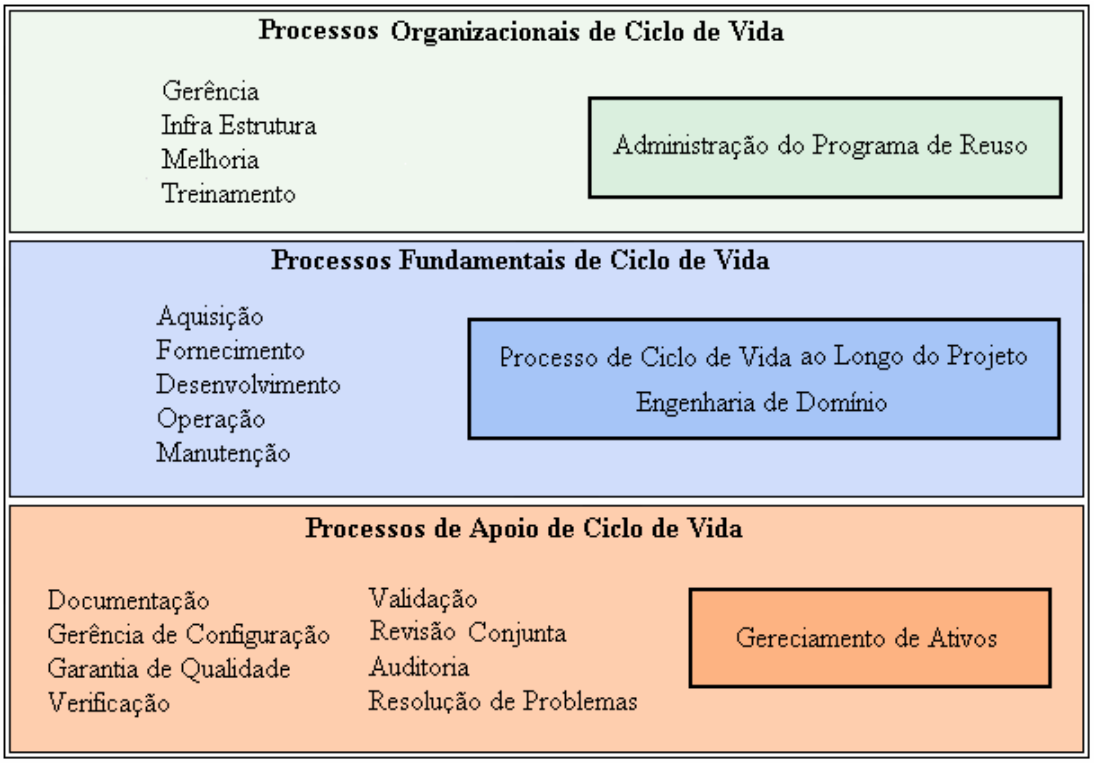


Figura 6 - Estrutura da norma IEEE Std 1517-1999 dentro da estrutura da norma IEEE/EIA-12207.0-1996

Os processos, atividades e tarefas da norma IEEE Std 1517-1999(R2004) foram divididos em quatro categorias de reuso:

I. Desenvolvimento, operação e manutenção de produtos de software com ativos:

- A cada processo dos **Processos Fundamentais de Ciclo de Vida** foi incorporado um conjunto de atividades e tarefas para o desenvolvimento, operação e manutenção de um produto de software a partir de ativos reutilizáveis, conforme Figura 7.

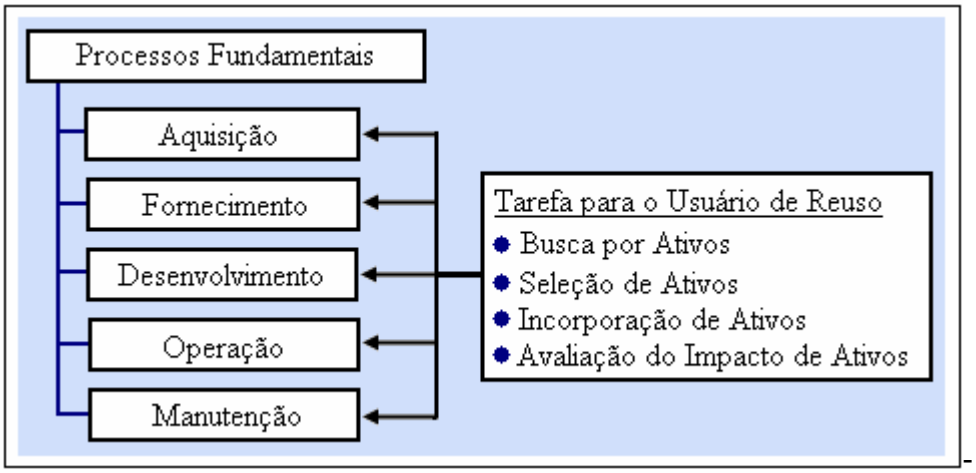


Figura 7 - Tarefas necessárias para os usuários de reuso, adicionadas a cada processo fundamental do Ciclo de Vida adaptado de Mcclure (2001, p.46).

II. Desenvolvimento e manutenção de ativos

- As atividades e as tarefas foram agrupadas em um processo denominado de **Processo de Engenharia de Domínio**, para atender o desenvolvimento e a manutenção de ativos. Este processo é definido como um processo de ciclo de vida ao longo do projeto (*cross-project life cycle process).* O Processo de Engenharia de Domínio atua ao longo do projeto, provendo ativos que podem ser usados por vários projetos. Estes ativos podem ser usados pelos **Processos Fundamentais do Ciclo de Vida** para adquirir, fornecer, desenvolver, operar e manter produtos de software.

- Conforme a Figura 8 o Processo de Engenharia de Domínio consiste de 5 (cinco) atividades (McClure, 2001):
  - **Implementação do processo**: desenvolver o plano da engenharia de domínio;
  - **Análise de domínio:** definir o domínio, o vocabulário do domínio e os modelos de domínio**;**
  - **Projeto do domínio:** produzir a arquitetura de domínio e as especificações de projeto dos ativos;
  - **Provisão de ativos:** desenvolver ou adquirir ativos do domínio;
  - **Manutenção de ativos:** manter os ativos do domínio**.**

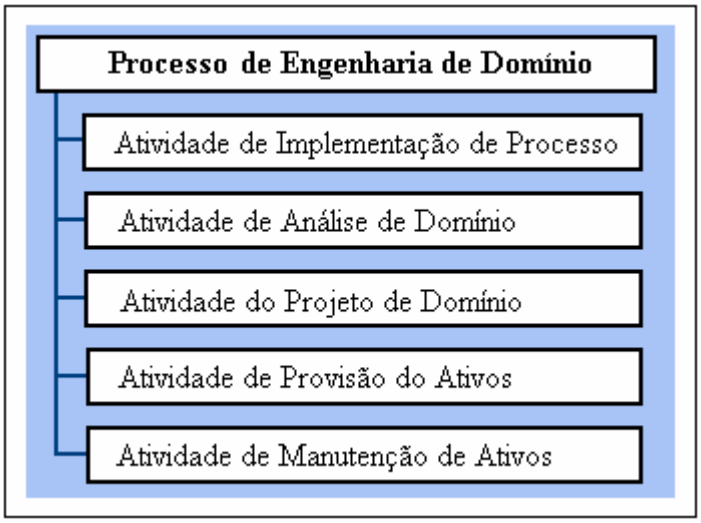


Figura 8 - Atividades do Processo da Engenharia de Domínio adaptado de McClure (2001, p. 250)

III. Administração da prática de reuso:

- As atividades e as tarefas foram agrupadas em um processo denominado de **Processo de Administração do Programa de Reuso** para planejar, estabelecer e gerenciar um programa de reuso;
- Este processo foi adicionado aos **Processos Organizacionais de Ciclo de Vida** e utiliza-se dos processos definidos na norma IEEE/EIA-12207.0-1996 (Gerenciamento, Infra Estrutura, Melhoria e Treinamento) para estabelecer uma estrutura de reuso integrado e para o desenvolvimento de software baseado em componentes.
- A Figura 9 apresenta o Processo de Administração do Programa de Reuso com os requisitos para um programa formal**,** incluindo as atividades (McCLURE, 2001):
  - Definir a estratégia de reuso da organização, incluindo seus propósitos, escopo, metas e objetivos.;
  - Avaliar a habilidade da organização para a prática do reuso sistemático;
  - Definir e implementar o plano de implementação do programa de reuso;

- Identificar os domínios em que existam oportunidades de reuso ou que se pretenda praticar reuso;
- Avaliar cada domínio para determinar seu potencial;
- Estabelecer uma estrutura de apoio organizacional e gerencial;
- Monitorar e executar o programa de reuso;
- Revisar e propor melhorias no programa de reuso.

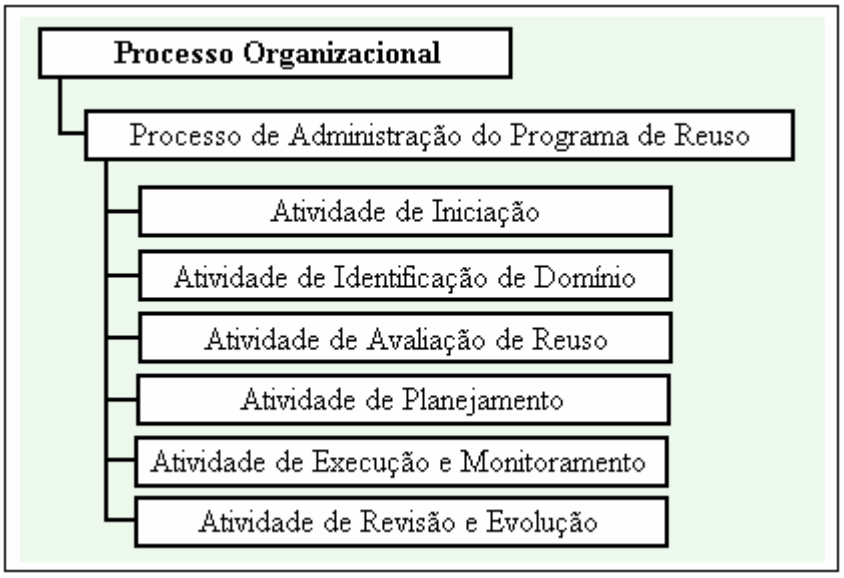


Figura 9 - Programa de Administração de Reuso como um Processo Organizacional do Ciclo de Vida adaptado de McClure (2001, p. 50)

IV. Administração de ativos

- As atividades e as tarefas foram agrupadas em um processo denominado de **Processo de Gerenciamento de Ativos** para apoiar os outros processos do ciclo de vida;

- Este processo foi adicionado aos **Processos de Apoio de Ciclo de Vida** para suportar os outros processos do ciclo de vida, entre eles, Desenvolvimento, Manutenção e Engenharia de Domínio, especificando as atividades necessárias para gerenciar a certificação, classificação, armazenamento, recuperação, controle de versão e controle de mudanças dos ativos;
- A Figura 10 mostra o **Processo de Gerenciamento de Ativos** com as atividades **(**McCLURE, 2001):
  - Desenvolver e implementar um plano de gerenciamento de ativos;
  - Implementar e manter um mecanismo para armazenar e recuperar um ativo, como por exemplo, uma biblioteca de reuso;
  - Desenvolver e manter um esquema de classificação de ativos e processos de certificação;
  - Avaliar e homologar novos ativos, atualizações e novas versões através de mecanismos de armazenamento e recuperação de ativos;
  - Gerenciar o armazenamento de ativos, reportando problemas e soluções.

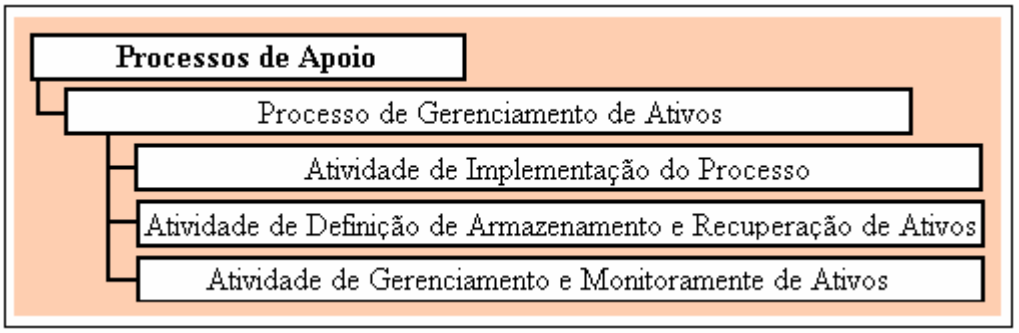


Figura 10 - Gerenciamento de Ativos como um Processo de Apoio de Ciclo de Vida adaptado de MCCLURE (2001, p. 52)

2.3.3 Processo RUP – Rational Unified Process

O *Rational Unified Process*® (também chamado de processo RUP®) também descreve uma estrutura comum de processos para cobrir o ciclo de vida de um software. Desenvolvido pela *Rational Software Corporation* e atualmente de propriedade da empresa IBM (*International Business Machine*), ele oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades necessárias ao desenvolvimento de software. O objetivo do RUP é garantir a produção de software de alta qualidade que atenda às necessidades dos usuários dentro de um cronograma e de um orçamento previsíveis (RUP, 2001).

A Figura 11 mostra a arquitetura geral do RUP em que a ênfase varia através do tempo. Por exemplo, nas iterações iniciais, é dedicado um tempo maior aos requisitos. Nas iterações posteriores, é dedicado um tempo maior à implementação.

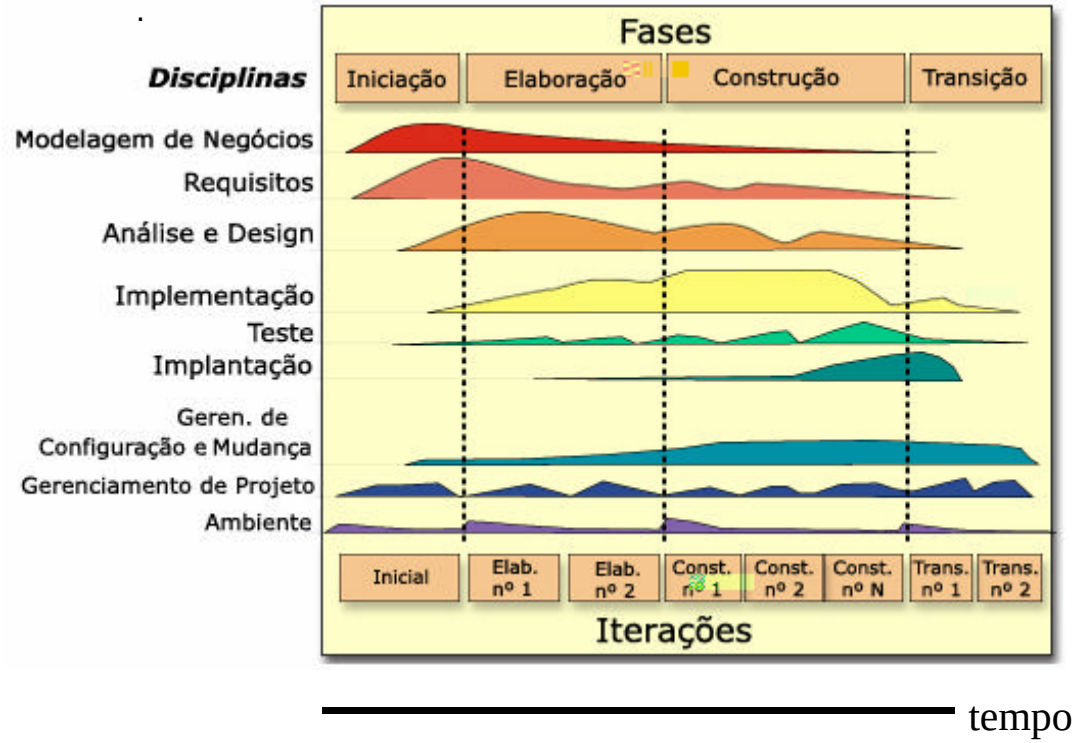


Figura 11 - Arquitetura Geral do RUP adaptada de RUP (2001)

O RUP tem duas dimensões:

- **Primeira Dimensão**: O eixo horizontal representa o tempo e mostra os aspectos do ciclo de vida do processo e é expresso em termos de fases, iterações e marcos.
- **Segunda Dimensão:** O eixo vertical representa as disciplinas que agrupam as atividades de maneira lógica, por Tvamsa

Esses componentes reutilizáveis formam a base de reutilização dentro de uma organização, aumentando a produtividade e a qualidade geral do *software* (RUP, 2001).

2.4 Base de Conhecimento

De acordo com TRINDADE (2006) “há várias definições para o termo conhecimento que acabam por resumi-lo à simplicidade objetiva e eficaz da informação tratada, compreendida e armazenada, aplicável e em plena aplicação, em finalidades diversas, segundo objetivos específicos e especificidades circunstanciais”.

Embora os termos “conhecimento” e “informação” sejam usados com freqüência como termos equivalentes, existe uma nítida distinção entre informação e conhecimento. A informação proporciona um novo ponto de vista para a interpretação de eventos ou objetos, o que torna visíveis significados antes invisíveis. Por isso, a informação é um meio ou material necessário para extrair e construir o conhecimento. Afeta o conhecimento acrescentando-lhe algo ou reestruturando-o (NONAKA & TAKEUCHI, 1996).

Ainda, segundo os autores, a informação pode ser vista de duas perspectivas: a informação “sintática” (ou o volume de informações) e a informação “semântica” (ou o significado). O aspecto semântico é mais importante para a criação do conhecimento, pois se concentra no significado transmitido.

Diversos autores teorizaram, além da criação, sobre a absorção, conservação, aplicação e disseminação deste que é considerado, tanto por profissionais de áreas de sistemas quanto de administração, um dos principais recursos estratégicos de gestão organizacional (TRINDADE, 2006).

Segundo TURBAN *et al.* (2005) a gestão do conhecimento é um processo que apóia as empresas a identificar, selecionar, organizar, disseminar, transferir e aplicar informações e experiências importantes que fazem parte da memória da empresa e que normalmente residem dentro das empresas de uma forma desestruturada.

Para que a empresa capacite-se a gerenciar o conhecimento, é necessário estruturar a organização deste ativo. Uma das formas de organizar o conhecimento refere-se ao conceito de taxonomia, uma abordagem para classificar o conhecimento a partir da combinação e esquematização de um conjunto de informações que formam seu conteúdo. (TRINDADE, 2006).

PFLEEGER (1999) *apud* TRINDADE (2006) demonstra na Figura 12 que o conhecimento, uma vez descrito, possui três níveis de informações organizacionais: sua classificação (os quais devem ser configurados conforme o negócio ou a empresa), sua área de aplicação (com possíveis sub-classificações de áreas necessárias) e sua composição (também reconfigurável segundo a aplicação).

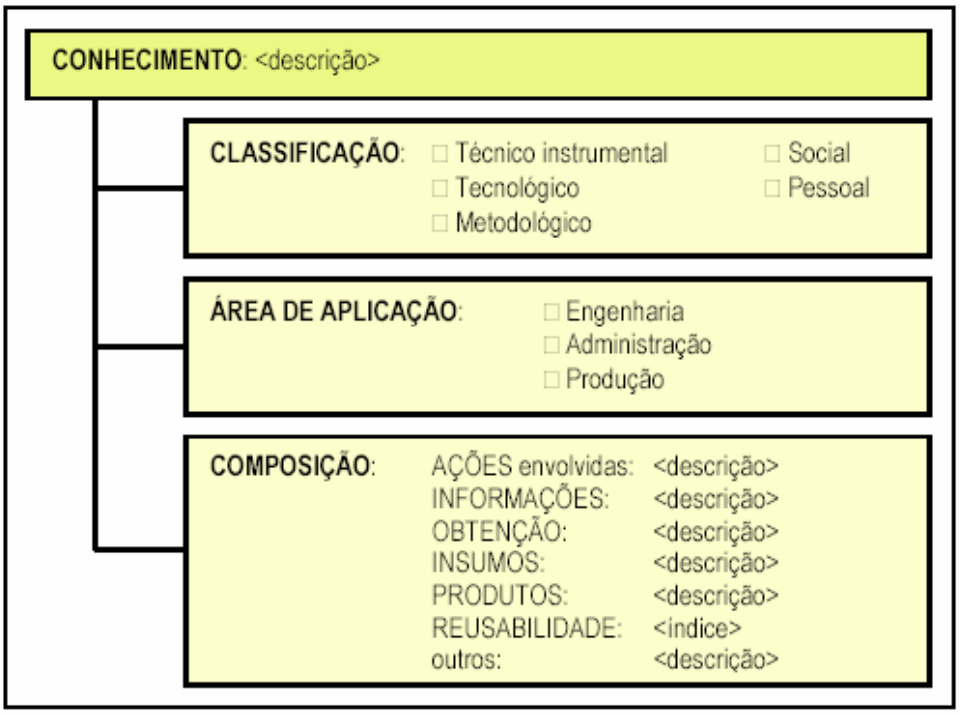


Figura 12 – Organização do conteúdo do conhecimento adaptado de PFLEEGER (1999, p.38) por TRINDADE (2006)

O conhecimento obtido deve ser compartilhado e disperso de forma ampla dentro da empresa, armazenado como parte de uma base de conhecimentos e utilizado pelos envolvidos no desenvolvimento de novas tecnologias e produtos (NONAKA & TAKEUCHI, 1996; TURBAN *et al.,* 2005; TRINDADE, 2006).

Implementar uma base de conhecimento sobre componentes de software reutilizáveis implica na formulação de um processo para criar, classificar, documentar, armazenar e disseminar o conhecimento sobre estes ativos, tornando tanto o processo quanto o próprio conhecimento gerenciáveis.

3 COMPONENTES DE SOFTWARE

Para PETERS & PEDRYCZ (2001), o terreno da engenharia de software é extremamente rico e variado. Uma das características dominantes neste terreno é a abordagem de engenharia utilizada no planejamento, conceituação e projeto de um software. Como exemplo de engenharia aplicada á produção de software, é recomendável comprar partes prontas de um sistema de software ou utilizar uma unidade de software já testada, que seja adaptável, portátil e .

Neste contexto, os componentes aparecem na Figura 13 como um dos principais propulsores do desenvolvimento de software.

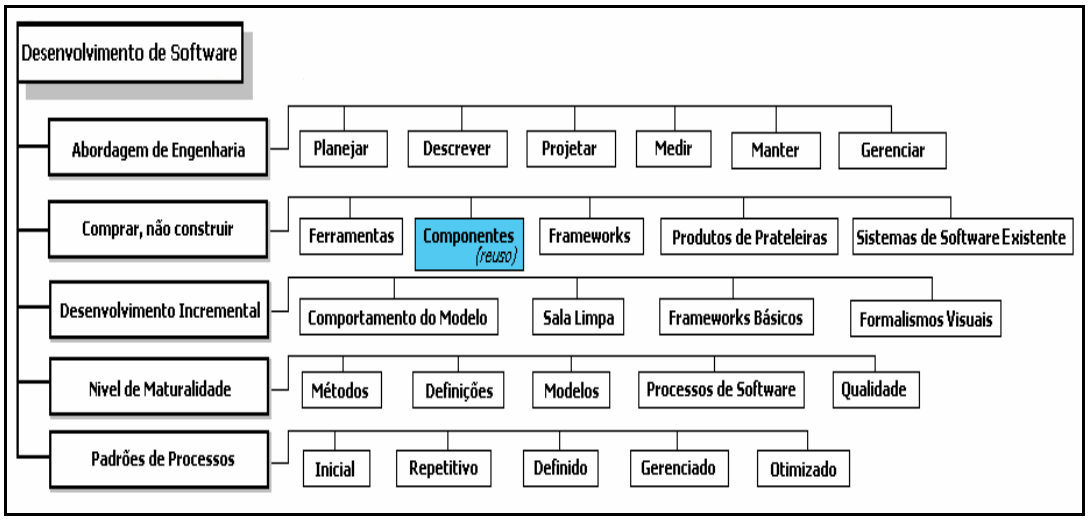


Figura 13 - Principais propulsores do desenvolvimento de software adaptado de PETERS (2001, p.3)

3.1 Conceitos e Definições

Em 1998, no 20th *International Conference on Software Engineering* foi realizado um workshop sobre o desenvolvimento de software baseado em componentes e foram levantadas diversas definições para componente (BROWN & WALLNAU, 1998):

- Componente é uma parte não trivial, quase independente, e substituível de um sistema, que cumpre uma função clara no contexto de uma arquitetura bem definida;
- Componente de software executável é um pacote montado dinamicamente em um ou mais programas, gerenciado como uma unidade e acessado através de *interfaces* documentadas;
- Componente de software é uma unidade de composição com *interfaces* especificadas contratualmente. Um componente de software pode ser utilizado independentemente e pode ser um elemento para composição de uma terceira parte;
- Componente de negócio representa a implementação em software de um conceito ou de um processo de negócio. Um componente de negócio consiste de artefatos de software necessários para expressar, implementar, e utilizar o conceito como um elemento de um sistema de negócio maior.

Os participantes do *workshop* notaram a importância de duas características adicionais para os conceitos de componentes de software:

- Relacionamento entre componentes e arquitetura de software;
- Relacionamento entre componentes e tecnologia de objetos.

A arquitetura de software fornece uma visão holística do sistema a ser construído, mapeando a estrutura e a organização dos componentes de software, suas propriedades e suas conexões. (PRESSMAN, 2006).

De acordo com o RUP (2001) "ao articular claramente os principais componentes e as interfaces críticas entre eles, uma arquitetura permite que você raciocine sobre a reutilização, tanto a reutilização interna que é a identificação das partes comuns, como a reutilização externa que é a incorporação de componentes desenvolvidos internamente ou adquiridos prontos para serem usados"

CHEESMAN & DANIELS (2001), ressaltam os princípios da tecnologia orientada a objetos como base, para conceituar um componente de software:

- Unificação de dados e funções: um objeto de software consiste de valores de dados (ou estados) e funções que processam aqueles dados. Esta colocação natural de dependências entre funções e dados melhora a coesão;
- Encapsulamento: um cliente (outro software) que utiliza um objeto de software está protegido da forma como os dados do objeto de software são armazenados ou como suas funções são implementadas. Isto é, o cliente depende da especificação do

objeto, mas não de sua implementação. Esta separação da descrição da implementação é chave para gerenciar as dependências entre os componentes de software e reduzir o acoplamento (dependência entre os componentes);

- Identidade: cada objeto de software tem uma identidade única, indiferentemente do estado.

O RUP (2001) acrescenta à definição de componente uma característica importante, a de ser substituível: "os componentes são grupos de código coesos, na forma de código fonte ou executável, com interfaces bem definidas e comportamentos que fornecem forte encapsulamento do conteúdo e são, portanto, substituíveis" .

Para BRAUDE (2005), componente é "um software utilizado sem alteração" e baseado na definição da OMG (*Object Management Group*) que acrescenta: "um componente representa uma parte física da implementação de um sistema, incluindo o código de software (fonte, binário ou executável) ou equivalentes, como *scripts* ou arquivos de comando".

SOMMERVILLE (2003) enfatiza duas importantes características de um componente :

- O componente é uma entidade executável independente, ou seja, o código fonte não está disponível, de modo que o componente não é compilado com outros componentes;

- Os componentes publicam suas interfaces e todas as interações são feitas por meio desta interface.

Os componentes de software devem estar disponíveis em uma biblioteca de reuso e/ou devem ser desenvolvidos por engenharia para atender os requisitos do cliente. Uma seqüência de atividades deve ser é aplicada quando um componente de software é identificado para reuso.

3.2 Especificação de Componente

Um componente é uma unidade de composição e deve ser especificado de tal forma, que é possível compô-lo com outros componentes e integrá-los nos sistemas de uma maneira previsível. Deve ter suas *interfaces* claramente documentadas e a sua implementação encapsulada (CRNKOVIC *et al.,* 2002).

Ainda, segundo CRNKOVIC *et al.* (2002) uma interface nomeia uma coleção de operações (funções ou serviços) e fornece as descrições e os protocolos destas operações. Esta separação torna possível substituir a parte de implementação sem mudar a interface e, adicionar novas interfaces (e implementações), sem mudar a implementação existente. Desta maneira, melhora-se a adaptabilidade do componente.

Enquanto uma interface especifica as operações de um componente, a especificação de um componente define o que será construído e quais unidades existirão em tempo de execução. A especificação de um componente define um conjunto de interfaces suportadas e as restrições que terão quando implementadas (CHEESMAN & DANIELS, 2001).

O Quadro 2 descreve as diferenças entre especificação de interface e especificação de componentes.

Quadro 2 - Comparação entre especificação de interface e especificação de componentes (CHEESMAN & DANIELS, 2001, p. 21)

Especificação de Interface versus Especificação de Componentes	
Interface	**Especificação de componentes**
Uma lista de operações	Uma lista de interfaces
Define o modelo de informação essencial para o tratamento do componente	Define os relacionamentos entre os modelos de informação de diferentes interfaces
Representa o contrato com o cliente	Representa o contrato com o implementador
Especifica como as operações afetam ou lidam com o modelo de informação	Define a unidade de implementação e execução
Descreve somente os efeitos locais	Especifica como as operações devem ser implementadas em termos de uso com outras interfaces

Para que as especificações sejam formalmente acordadas, deve ser definido um conjunto de regras que especificam obrigações mútuas, denominado de contrato. No caso de contrato de componentes, o seu conceito está relacionado a aspectos relativos à confiabilidade e ao comportamento inesperado no caso de falha ou erro (ROSSI, 2004).

Para CHEESMAN & DANIELS (2001) existem dois tipos diferentes de contrato: o contrato de uso e o contrato de realização. O contrato de uso está relacionado com a execução, e o contrato de realização com o projeto.

O contrato de uso tem a finalidade de descrever o relacionamento entre os objetos da interface de componente e um cliente e é especificado

na forma de uma interface. Desta forma a especificação de uma interface deve incluir:

- Operações: a lista de operações que a interface disponibiliza, incluindo suas assinaturas e definições;
- Modelo de informação: a definição abstrata de qualquer informação ou estado que atende uma requisição de um cliente através de objetos suportados pela interface e qualquer restrição.

O contrato de realização é o contrato entre a especificação de componente e sua implementação e deve estar de conformidade pela pessoa que está criando a implementação do componente (realizador ou programador). Desta forma, o contrato de realização é incorporado na própria especificação de componentes.

BRAUDE (2005) apresenta o conceito de manifesto para especificar um componente. Manifesto é uma lista com informações que identificam o conteúdo de um componente, entre elas, identificação do componente, autoria do componente, lista de arquivos ou classes que compõem este componente, outros componentes que este utiliza, informações de criptografia, meios de verificar se todas as partes do componente estão presentes e número da versão.

A especificação do componente é apoiada pela sua documentação, com as informações necessárias para sua utilização.

3.3 Documentação de Componentes

No processo de desenvolvimento de sistemas através da utilização de componentes é necessário descrever, compreender e identificar como integrá-los com outros componentes. Uma forma de representar e documentar este conjunto de informações é através de uma linguagem padrão de modelagem como a UML (*Unified Modeling Language*), que define como modelar componentes através de seu metamodelo. (HOPKINS, 2000).

Metamodelo é um modelo que define a linguagem para expressar um modelo (RUP, 2001).

A UML é uma linguagem para especificação, visualização, construção e documentação de sistemas de softwares, bem como para modelagem de negócios e outros sistemas não ligados a software (FOWLER & SCOTT, 2000).

A UML suporta a modelagem de componentes, desde a modelagem lógica, por meio do diagrama de classes, passando pela modelagem física, por meio do diagrama de componentes e diagrama de operação ou *deployment diagram*. Desta forma, conforme HOPKINS (2000), a UML permite a rastreabilidade do componente por meio destes diagramas, desde a especificação lógica, passando pela definição do componente, até a execução.

Embora o UML seja uma linguagem de modelagem e não um processo de software, ela fornece um conjunto de notações (principalmente gráficas) que podem ser usadas nas várias fases no ciclo de vida de um software (KOBRYN, 2000).

Os componentes na UML são modelados tipicamente nos diagramas de implementação, entre eles, o diagrama de componentes e o diagrama de operação (*deployment diagram*).

Um diagrama de componentes mostra a organização e as dependências dos componentes, e um diagrama de operação mostra como os componente e as instâncias de uma classe são acessados em nós computacionais.

3.4 Classificação e Recuperação de Componentes

A classificação de componentes é uma atividade para organizar e catalogar um conjunto de componentes de acordo com um esquema determinado.

A maioria dos esquemas de classificação para componentes de software enquadra-se em três categorias (PRESSMAN, 2006; SAMETINGER, 1997):

- **Classificação enumerada**: os componentes são descritos por uma estrutura hierárquica em que as categorias e os vários níveis de subcategorias de componentes de software são definidos. A estrutura hierárquica de um esquema de classificação enumerada é fácil de entender e usar. No entanto, apresenta uma estrutura

inflexível, em que todas as categorias devem ser inicialmente definidas. O sistema decimal de Dewey (DEWEY, 1979 *apud* SAMETINGER, 1997) é um exemplo de aplicação deste método aplicado na organização de uma biblioteca. Os livros ou outras publicações são classificados por meio de classes designadas por um número de três dígitos e as subdivisões são apresentadas por números depois do ponto decimal;

- **Classificação facetada**: Na classificação facetada cada componente é descrito por uma lista ordenada de características chamada de facetas. Uma faceta é um tipo de descritor que ajuda a identificar o componente, e uma coleção de facetas permite identificar diversas características ao mesmo tempo. Normalmente, o conjunto de facetas é limitado a não mais que sete ou oito facetas;
- **Classificação de valores de atributos**: Um conjunto de atributos é definido para todos os componentes. São atribuídos valores a estes atributos, semelhante à classificação facetada. A diferença está no número de atributos que não é limitado, não é atribuída prioridade aos atributos e a busca e recuperação de um componente requerem uma especificação exata dos valores associados aos atributos.

O método de classificação por facetas, segundo SAMETINGER (1997) apresenta maior flexibilidade, permitindo a expansão dos elementos

do esquema de classificação, relações complexas podem ser criadas através da combinação de facetas e termos, sendo que as facetas podem ser alteradas individualmente, sem afetar as demais.

O método de classificação facetada, proposto por Prieto-Diaz (PRIETO-DÍAZ, 1991), é baseado na análise de um assunto e um conjunto de características descritivas básicas é identificado. Estas características, ou facetas, são ordenadas por importância e conectadas a um componente. Uma faceta pode descrever a função que o componente executa, os dados que são manipulados, o contexto no qual são aplicados ou qualquer outra característica. Desta forma, o assunto em um domínio é descrito pela síntese destas facetas básicas. As facetas são consideradas como perspectivas, visões ou dimensões de um domínio de aplicação em particular (PRIETO-DÍAZ, 1991).

PRIETO-DÍAZ (1993) apresenta uma classificação para os tipos de reuso, que são divididos em pelo menos seis perspectivas ou facetas, que são: substância, escopo, modo, técnica, intenção e produto. O Quadro 3 apresenta as perspectivas propostas e os artefatos relacionados nesta classificação.

Quadro 3 - Aspectos de reutilização (PRIETO-DIAZ, 1993, p. 52)

Facetas de Reuso					
Substância	**Escopo**	**Modo**	**Técnica**	**Intenção**	**Produto**
Idéias, conceitos	Vertical	Planejado, sistemático	Por composição	Caixa-preta, original	Código-fonte
Artefatos, componentes	Horizontal	Ad-hoc, oportunista	Por geração	Caixa-branca, modificada	Projeto
Procedimentos, habilidades e experiência					Especificação
					Objetos
					Testes
					Arquiteturas

- **Perspectiva de substância:** define a essência dos itens a serem reutilizados, que podem ser idéias ou conceitos, artefatos ou componentes e procedimentos ou habilidades. Entende-se por reuso de idéias ou conceitos, a reutilização de conceitos formais, como por exemplo, reutilização de soluções genéricas para uma classe de problemas. O reuso de artefatos ou componentes refere-se à reutilização de partes de um sistema ou de um projeto em um novo sistema. O reuso de procedimentos ou habilidades é focado na formalização e encapsulamento de procedimentos de desenvolvimento de software, ou seja, na criação de coleções de processos que podem ser conectados para instanciar processos novos e mais complexos;
- **Perspectiva de escopo:** estabelece a forma como a reutilização ocorre dentro de uma família de sistemas ou entre famílias de sistemas, ou seja, no domínio. A perspectiva do escopo pode ser do tipo vertical ou horizontal. A reutilização vertical estabelece o

reuso dentro do mesmo domínio ou de uma área de aplicação. O objetivo é gerar modelos genéricos para famílias de sistemas, os quais podem ser utilizados como gabaritos na criação de novos sistemas. O foco é a análise e a modelagem do domínio, pois através destas atividades, torna-se possível identificar os modelos genéricos para famílias de sistemas. A reutilização horizontal reutiliza partes genéricas em diferentes aplicações, ou seja, o objeto de reuso pode ser utilizado em diferentes domínios.

- **Perspectiva de modo:** define como o reuso é conduzido, que pode ser *ad hoc* ou sistemático. A reutilização *ad hoc* ou oportunista é o modo mais comum. A sua principal característica é a informalidade e os objetos de reuso são selecionados de bibliotecas genéricas. A reutilização é conduzida pelo indivíduo e não pelo projeto, e não há processos definidos para que ela ocorra. Na reutilização sistemática ou planejada, são definidos e seguidos as diretrizes e os procedimentos e, o desempenho é avaliado através de métricas. Existe a necessidade do comprometimento da gerência e um grande investimento deve ser feito para implantar este tipo de reutilização.

- **Perspectiva de técnica** define a abordagem utilizada para implementar o reuso, que pode ser gerativa ou composicional. A abordagem gerativa ou reutilização por geração consiste na reutilização no nível de especificação, através do emprego de

geradores de código ou de aplicações. Assim, uma vez criados os mecanismos de geração, o esforço para criação de novas aplicações é reduzido. A abordagem composicional ou reutilização por composição é a utilização de componentes existentes como blocos para construção de um novo sistema. Segundo PFLEEGER (2004), nesta abordagem os componentes são armazenados em uma biblioteca (ou repositório de reutilização), classificados ou catalogados e, um sistema de recuperação deve ser utilizado para procurar e selecionar os componentes a serem utilizados. O código-fonte é o artefato de reutilização mais comum neste tipo de abordagem.

- **Perspectiva de intenção:** refere-se ao aspecto de alterabilidade do componente de reuso e pode ser caixa-branca ou caixa-preta. Entende-se por reutilização caixa-branca, quando o componente reutilizado é modificado e adaptado. A principal dificuldade neste tipo de reutilização é o controle de alterações dos componentes, que implica em um controle de versões mais formalizado. Na reutilização caixa-preta, o componente de software é reutilizado sem que qualquer alteração seja feita. Os componentes são encapsulados e suas funcionalidades são acessadas por meio de interfaces padronizadas. A principal vantagem neste tipo de reutilização é a garantia de maior qualidade e confiabilidade. Porém, o custo para a criação de componentes reutilizáveis é

mais alto do que na reutilização caixa-branca, uma vez que a complexidade de construção deste componente é maior.

- **Perspectiva de produto** estabelece os produtos de um projeto que podem ser reutilizados:
  - Código-fonte: é a prática de reutilização mais comum;
  - *Design*: utiliza as especificações para auxiliar na identificação das partes do projeto que podem ser reutilizadas. *Design patterns* ou padrões de projetos, apresentados por ERICH GAMMA em 1995 representam a principal forma de reuso de *design* (GAMMA *et al.*, 1995). Estes padrões de projeto apresentam soluções para determinado tipo de problema, que podem ser implementados de um modo específico (SAMETINGER, 1997);
  - Especificação: quando uma especificação torna-se disponível para reuso, ela é empacotada (encapsulada) com suas implementações no nível de *design* e código;
  - Objetos: permite a reutilização dentro de um mesmo sistema ou entre sistemas diferentes, devido aos conceitos de abstração, encapsulamento e herança;
  - Textos: permite a reutilização dos documentos do projeto de sistema. A documentação é uma importante parte de um

sistema de software e pode estar relacionada a documentação de um produto ou de um processo. A documentação de um produto descreve como utilizar um sistema (uma descrição de *interface* do usuário, uma descrição funcional, um manual de referência) e como este produto é implementado (arquitetura de um sistema, implementações de componentes, entre outros). A documentação de um processo descreve o processo de criação (planejamento, estimativas, programação) (SAMETINGER, 1997);

- Arquitetura: é um elemento fundamental para composição de um sistema na abordagem composicional. Uma arquitetura de software é uma estrutura global de um sistema de software com seus principais subsistemas, incluindo a especificação destes subsistemas, seus inter-relacionamentos e a tecnologia apropriada (SAMETINGER, 1997; CHEESMAN & DANIELS, 2001). Uma definição específica para arquitetura de componentes, segundo CHEESMAN & DANIELS (2001) é uma coleção de componentes, em um nível mais alto de abstração, suas interações e suas dependências de comportamento.

"Palavras-chave" são atribuídas ao conjunto de facetas para cada componente. Quando uma consulta é efetuada à biblioteca para buscar

possíveis componentes para um projeto, uma lista de valores é especificada e a biblioteca é percorrida para encontrar coincidências. (PRESSMAN, 2006).

Conforme PRIETO-DÍAZ (1991), as “palavras-chave” podem ser construídas através de um “vocabulário controlado”. Estas “palavras-chave” são derivadas e definidas por especialistas e descrevem ou representam os conceitos relevantes para o domínio da aplicação. Uma lista consistente de termos (*Thesaurus*) para cada faceta irá prover um vocabulário padrão, tanto para a biblioteca como para os usuários.

3.5 Qualificação e Qualidade de componente

Para o IEEE *Computer Society* qualificação é um processo para determinar se um sistema ou um componente está adequado para uso operacional (IEEE Std 610.12-1990).

Segundo PRESSMAN (2006), a qualificação procura assegurar que um componente candidato realiza as funções solicitadas, que atenda os padrões arquiteturais adotados, e que possua as características de qualidade exigidas para a aplicação, como por exemplo, desempenho, confiabilidade e usabilidade.

O nível de qualidade dos produtos deve atender aos requisitos do projeto por meio de avaliação que certifiquem e documentem estes produtos (MORISIO *et al.*, 2000).

Conforme AZUMA (1996) *apud* COSTA (2003), a avaliação é a chave do sucesso tanto para o desenvolvimento quanto para a aquisição de um bom software. A avaliação é necessária em vários pontos do ciclo de vida de software para vários artefatos, dependendo de um propósito específico. Em geral, o propósito da avaliação é julgar quanto um artefato é bom em relação aos seus objetivos específicos, o que inclui:

- Selecionar alguma coisa de duas ou mais alternativas;
- Estimar os valores presumíveis de um artefato;
- Avaliar o efeito do artefato quando ele é usado;
- Obter informações nos produtos intermediários para a obtenção do artefato ou processo para controlar e gerenciar o processo.

Para avaliar a qualidade de um produto por algum método quantitativo, é necessário um conjunto de características de qualidade que descreva o produto e forme a base para a avaliação. A norma NBR ISO/IEC 9126 -1 (2003), sob o título “Engenharia de Software – Qualidade de Produto – Parte 1: Modelo de Qualidade” define essas características de qualidade para produtos de software.

A norma NBR ISO/IEC 9126-1 (2003) estabelece ainda que essas características de qualidade de software sejam refinadas em múltiplos níveis de subcaracterísticas, conforme o Quadro 4.

Quadro 4 - Características da Qualidade adaptado da norma NBR ISO/IEC 9126-(2003)

Características da Qualidade	Subcaracterísticas
FUNCIONALIDADE	**Adequação, Acurácia (precisão), Interoperabilidade Segurança de acesso e Conformidade com a funcionalidade**
CONFIABILIDADE	**Maturidade, Tolerância a falhas, Recuperabilidade e Conformidade com a confiabilidade**
USABILIDADE	**Entendimento, Aprendizagem, Operabilidade e Conformidade com a usabilidade**
EFICIÊNCIA	**Comportamento no tempo, Comportamento dos recursos e Conformidade com a eficiência**:
MANUTENIBILIDADE	**Analisabilidade, Modificabilidade, Estabilidade Testabilidade e Conformidade com a manutenibilidade**
PORTABLIDADE	**Adaptabilidade, Capacidade para ser instalado, Coexistência, Substituibilidade e Conformidade com a portabilidade**:.

O modelo proposto pela norma NBR ISO/IEC 9126 -1 (2003) pode ser utilizado para especificar requisitos funcionais e não funcionais.

De acordo com SIMÃO (2002), “a qualidade de componentes de software é de fundamental importância para o sucesso do CBD, para a construção de aplicações mais rápidas, com menor custo e maior qualidade”. Mas, apesar de algumas iniciativas, não existe um modelo de qualidade para componentes que defina propriedades de qualidade para componentes de forma geral, que seja organizado de forma padrão, extenso o suficiente para abranger vários aspectos da qualidade, e que possa ser instanciado para diversos tipos de componentes.

A adoção de um ambiente de desenvolvimento de software baseado em componentes implica em desenvolver ou adquirir componentes confiáveis e disponíveis para reutilização. Para PFLEEGER (2004) existem

quatro características a serem verificadas nos componentes que serão reutilizados:

1. O Componente realiza a função ou fornece os dados solicitados?
2. Se uma pequena modificação for exigida, esta será menor do que construir o componente a partir do zero?
3. O componente é bem documentado, de modo a entendê-lo sem a necessidade de verificar a implementação linha por linha?
4. Existe um registro dos testes do componente e do histórico da revisão para assegurar que o componente não contém nenhuma falha?

E, acrescenta que deve ser avaliada a quantidade de código que precisará ser escrito para que o sistema possa realizar a interface com os componentes.

A partir dos conceitos e definições apresentadas no capítulo, um componente de software deve ser considerado como um ativo da organização e o reuso deste ativo é essencial para agilizar o desenvolvimento de software. O desenvolvimento baseado no reuso é uma abordagem de desenvolvimento que propõe maximizar o uso destes ativos. Contudo, é necessário formalizar um processo para suportar o uso de componentes reutilizáveis.

4 DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES

O desenvolvimento baseado em componentes – *Component Based Development* (CBD) ou a Engenharia de Software Baseada em Componentes – *Component-Based Software Engineering* (CBSE) consolidou-se no final de 1990 com uma abordagem baseada no reuso, motivada pela frustração de que o desenvolvimento orientado a objetos não tinha conduzido a um efetivo reuso, como originalmente foi sugerido (SOMMERVILLE, 2003).

Segundo PRESSMAN (2006), o CBSE é semelhante á engenharia de software convencional. O processo inicia-se com estabelecimento dos requisitos do sistema utilizando as técnicas tradicionais de elicitação de requisitos. Em seguida, o projeto arquitetural é estabelecido. Entretanto, antes de continuar detalhando as atividades de projeto, procura-se identificar o conjunto de requisitos que pode ser atendido pela composição de componentes pré-existentes, através de três questões que são aplicadas a cada requisito:

- Existem componentes disponíveis no mercado, componentes COTS, para implementar o requisito?
- Existem componentes reutilizáveis internos para implementar o requisito?
- As interfaces dos componentes disponíveis são compatíveis com a arquitetura do sistema a ser construído?

Para os requisitos que não foram atendidos por componentes pré-fabricados ou por componentes internos, emprega-se a engenharia de software tradicional para desenvolver novos componentes para satisfazer os requisitos.

A atividade de construção de aplicações com componentes ocorre em paralelo com a atividade de construção de componentes.

SOMMERVILLE (2003) ressalta que o CBD pode estar associado a um processo de desenvolvimento de software, agregando atividades específicas para reuso.

O processo RUP, introduzido no capítulo 2, seção 2.3.3, incorpora às suas disciplinas as atividades para suportar o desenvolvimento de software baseado em componentes (RUP, 2001) :

- A abordagem iterativa permite identificar componentes progressivamente e decidir quais desenvolver, quais reutilizar e quais comprar;
- O foco na arquitetura de software permite montar a estrutura, os componentes e como eles se integram, incluindo os padrões e os mecanismos fundamentais, através dos quais eles interagem. No RUP, a arquitetura de um sistema de software (em um determinado ponto) é a organização ou a estrutura dos componentes significativos do sistema que interagem por meio de interfaces;

- Conceitos como pacotes, subsistemas e camadas são utilizados durante a disciplina **Análise e *Design*** para organizar componentes e especificar interfaces. Na Figura 14, é apresentado o fluxo de atividades de análise e projeto, segundo a notação do RUP (RUP, 2001).

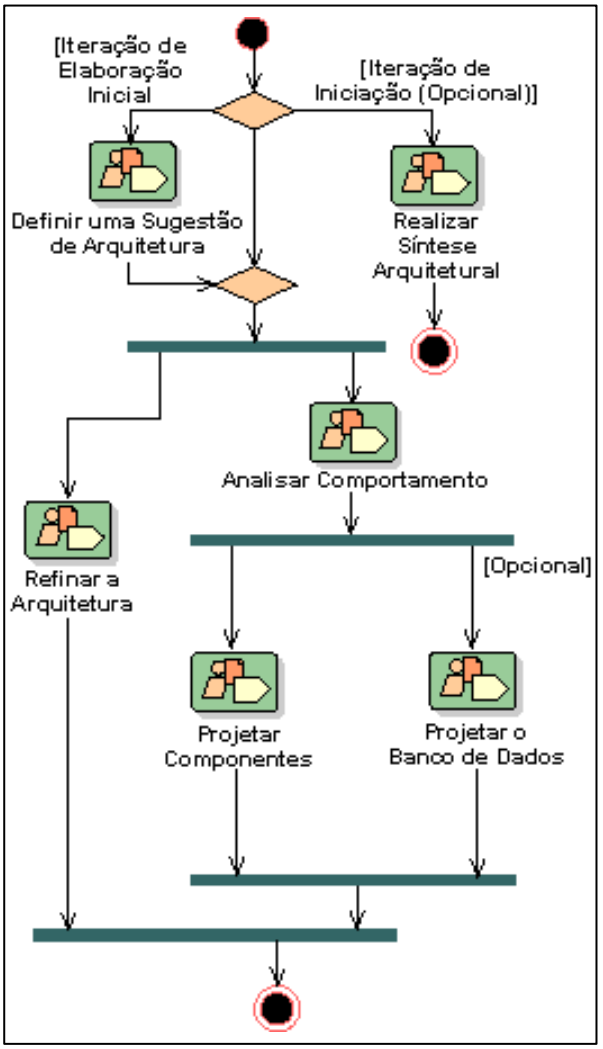


Figura 14 - Fluxo de Trabalho da disciplina Análise e *Design* (RUP, 2001)

- Os testes são primeiramente organizados em componentes e, em seguida, em conjuntos maiores de componentes integrados,

durante a disciplina Implementação. A Figura 15, apresenta o fluxo das atividades de implementação, segundo a notação do RUP (RUP, 2001).

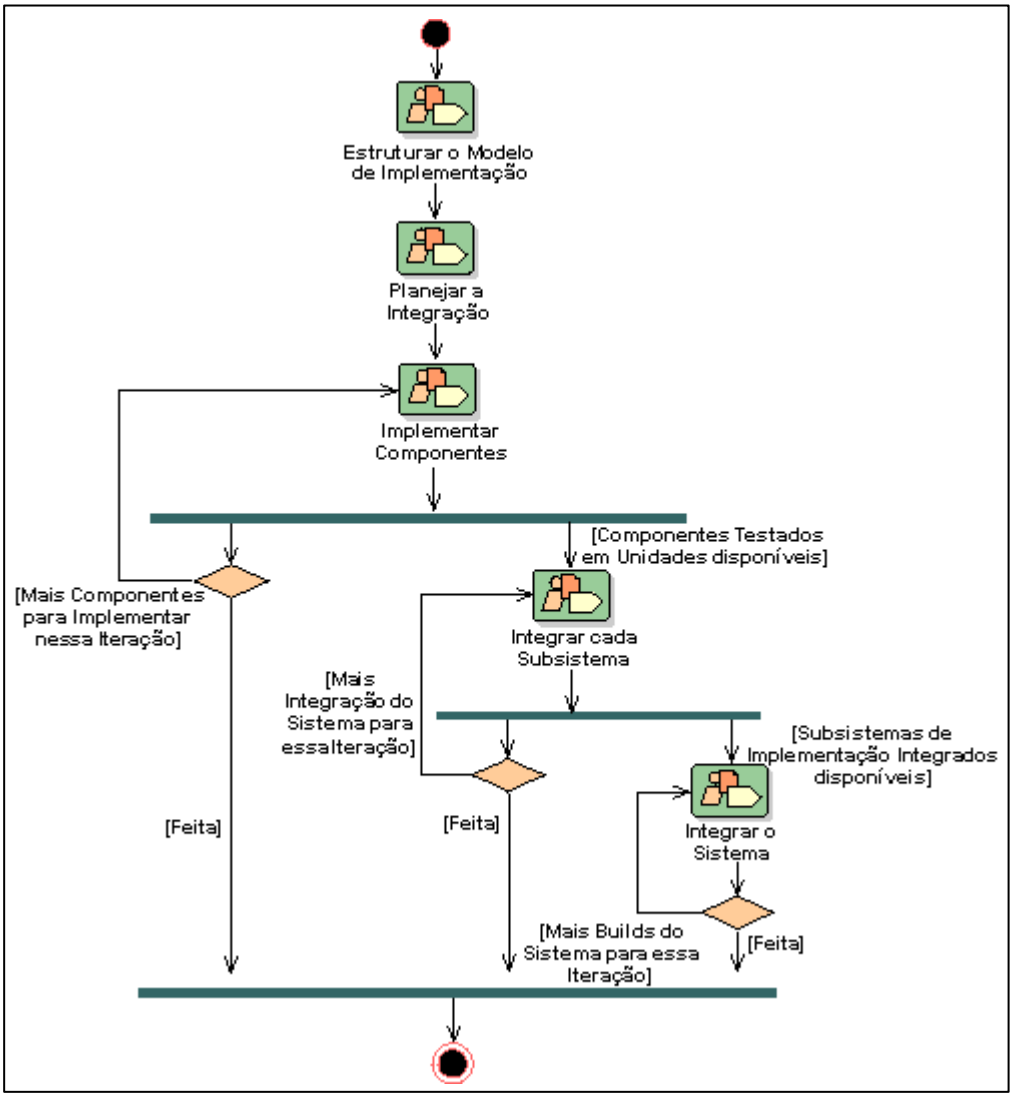


Figura 15 - Fluxo de Trabalho da disciplina Implementação (RUP, 2001)

CHEESMAN & DANIELS (2001), apresentam um processo “simples”, para especificar um software utilizando-se de componentes reutilizáveis,

baseado no RUP. Este processo segue um ciclo de vida iterativo e incremental e emprega a UML como linguagem de especificação, desde as fases de iniciais do desenvolvimento, através dos diagramas de casos de uso (*Use Case Diagrams*), até as fases finais de projeto com diagramas de classes, de seqüência, componentes, entre outros.

O conceito de disciplina, assim como o processo RUP também é utilizado. Cada disciplina define "uma seqüência de atividades que produz um resultado de valor observável". São definidas as disciplinas de Requisitos, Especificação, Provisão, Montagem, Teste e Implantação. As disciplinas de Requisitos, Teste e Implantação correspondem aos especificados no RUP. As disciplinas de Análise e Projeto e Implementação foram substituídas por Especificação, Provisão e Montagem.

A Figura 16 apresenta uma visão geral do processo. Resumidamente, a disciplina de Especificação recebe os artefatos gerados em Requisitos. A Especificação define as especificações das interfaces e como os componentes interagem com outros componentes.

Com a definição das especificações dos componentes, a disciplina de Provisão determina que componentes devem ser construídos e os que devem ser reutilizados ou comprados. O objetivo é garantir que os componentes necessários estejam disponíveis: ou pela construção total de componentes, ou pela aquisição de terceiros, ou pela adaptação de componentes existentes. A disciplina de Montagem é responsável pela

combinação dos componentes e a construção de interfaces de usuário, em uma aplicação que encontre as necessidades inicialmente levantadas.

A disciplina de Teste consiste na verificação do atendimento dos requisitos funcionais e não funcionais levantados em Requisitos. A aplicação é disponibilizada na Implantação.

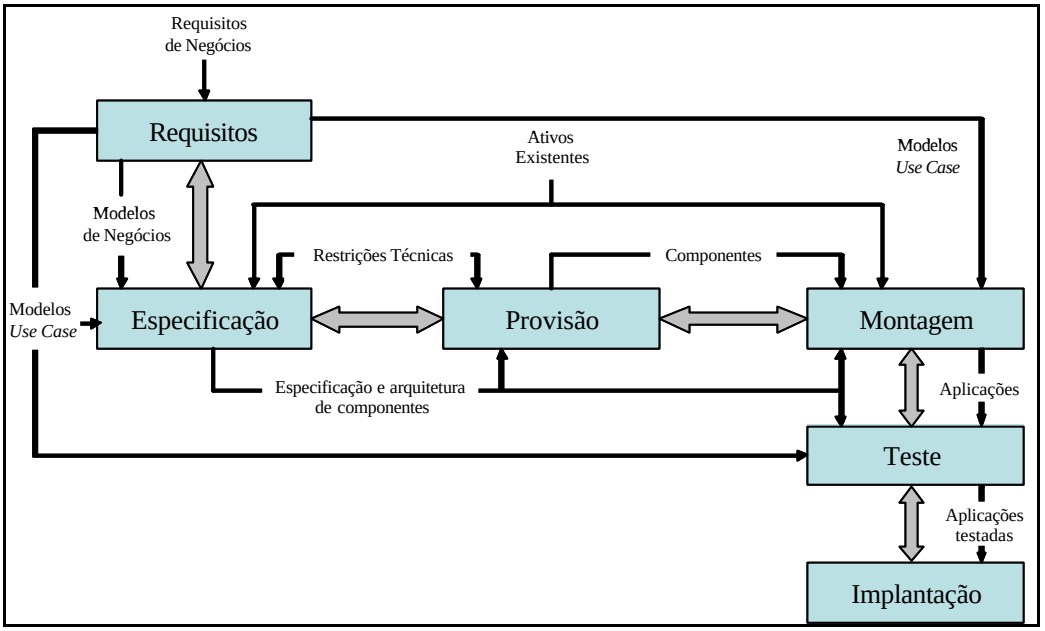


Figura 16 - Visão geral das disciplinas do processo de desenvolvimento proposto por CHEESMAN & DANIELS (2001)

Da mesma forma, as normas introduzidas no capítulo 2, seção 2.3, com os processos de ciclo de vida de software e suas integrações (apresentadas na Figura 4, p. 42), também são referências para apoiar o CBSE.

HOPKINS (2000) ressalta dois aspectos importantes no desenvolvimento de um sistema baseado em componentes:

- Reutilização: habilidade para reutilizar componentes existentes para criar um sistema mais complexo, e
- Evolução? desenvolver um sistemas que é altamente "componentizado", torna um sistema mais fácil de manter.

Entretanto, segundo a autor, para atender os aspectos de reutilização e evolução devem existir:

- fornecedores de componentes aplicáveis e bem construídos que possam ser identificados, licenciados e facilmente usados;
- um modelo de componente (regras e convenções que guiam o desenvolvimento e a composição de componentes), que possa dar suporte a combinação e a interação de componentes; e
- um processo e arquitetura que suporte o ambiente de desenvolvimento baseado em componentes.

KRONIG, COSTA e SPÍNOLA (2005), apresentam na Figura 17 as principais atividades para apoiar o desenvolvimento de software baseado em componentes. Estas atividades estão relacionadas com a análise dos requisitos de negócio, o que permite identificar os componentes, passando pela definição e implementação desses componentes. Para garantir o reuso de softwares existentes e catalogados, evitando a construção de componentes semelhantes ou iguais, é utilizado o Repositório de Componentes ou Biblioteca de Componentes como um recurso de

armazenamento centralizado e organizado das informações relacionadas a um componente reutilizável.

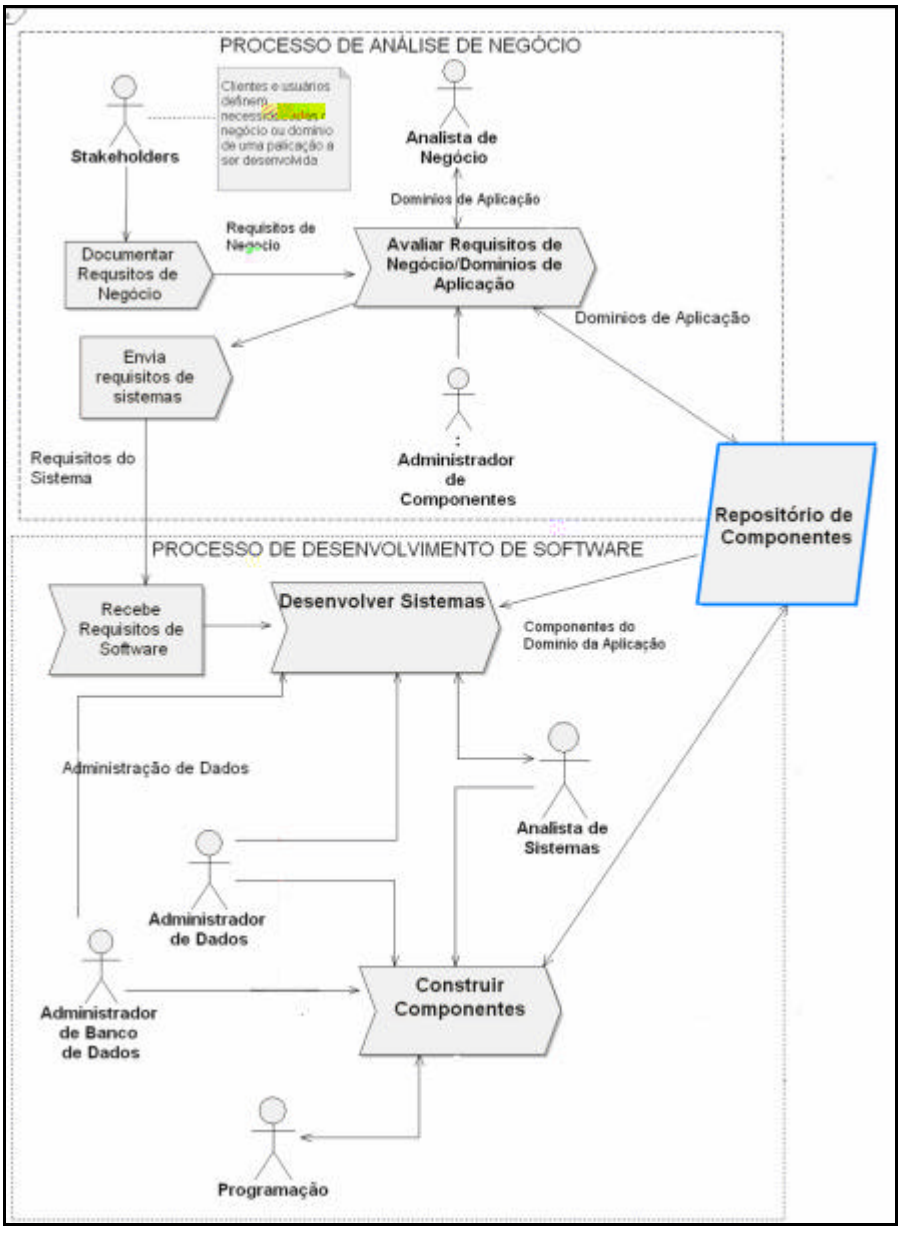


Figura 17 - Visão dos processos de desenvolvimento de software baseado em componentes (KRONIG, COSTA e SPÍNOLA, 2005)

De acordo com PRIETO-DIAZ (1991) deve existir um comprometimento gerencial e uma infra-estrutura efetiva para implementar um programa global de reuso:

- Primeiramente deve ser criado um grupo de suporte gerencial para prover e planejar as políticas e procedimentos para o reuso;
- Um sistema de biblioteca de fácil acesso e utilização, densamente populado; e
- Um grupo responsável por identificar, qualificar e certificar os componentes armazenados na biblioteca;
- Um grupo responsável pelo desenvolvimento, manutenção e atualização dos componentes;
- Um grupo de suporte para treinar e as apoiar as equipes envolvidas com o reuso (*reusers*).

A norma IEEE Std 1517-1999(R2004), introduzida no capítulo 2, seção 2.3.2, adiciona aos **Processos Organizacionais de Ciclo de Vida**, o **Processo de Administração do Programa de Reuso** para planejar, estabelecer e gerenciar um programa de reuso. Aos **Processos de Apoio de Ciclo de vida** é adicionado o **Processo de Gerenciamento de Ativos** para administrar a certificação, classificação, armazenamento, recuperação, controle de versões e mudanças dos ativos.

KRONIG, COSTA e SPÍNOLA (2005), na Figura 18, propõem um processo de Administração de Componentes como apoio ao desenvolvimento de software baseado em componentes cuja missão é promover o reuso de componentes de software.

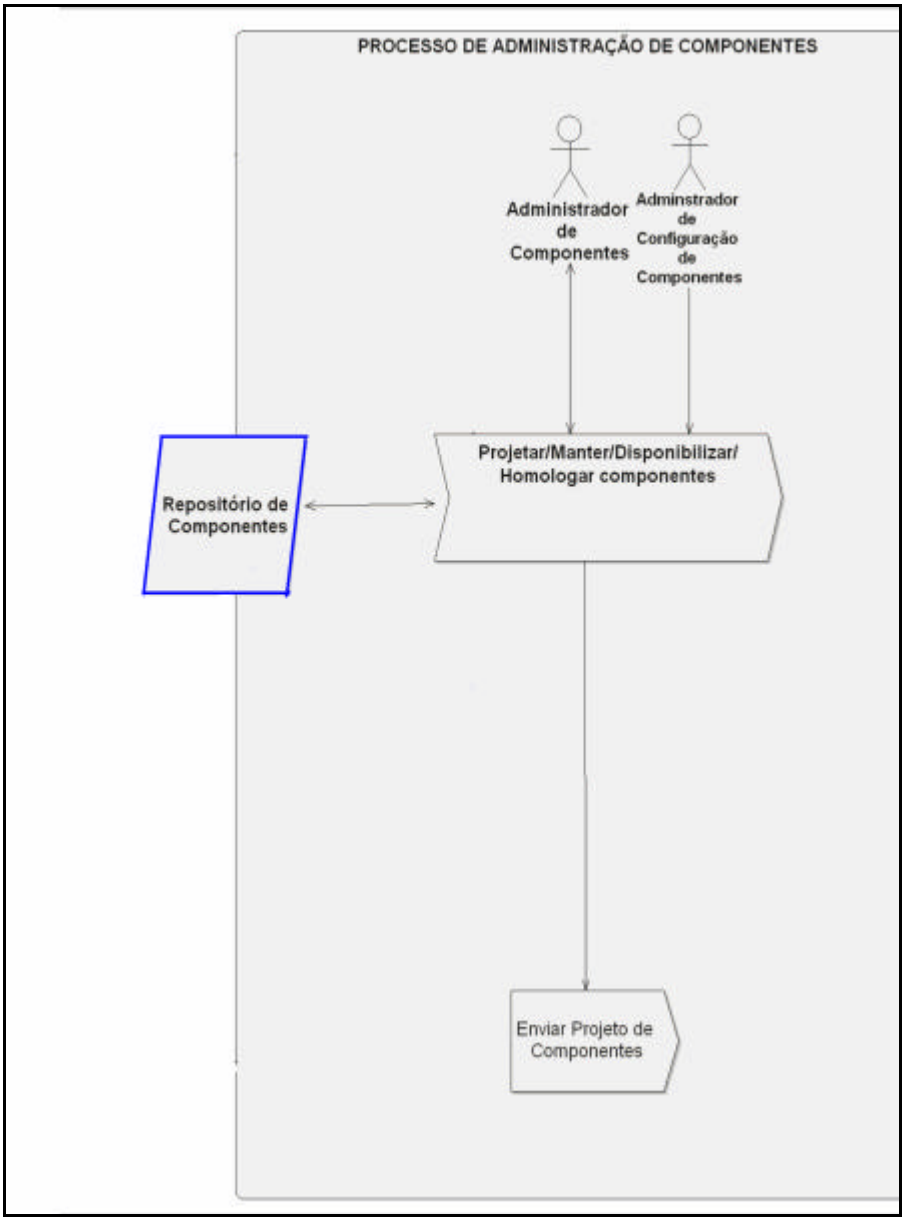


Figura 18 - Visão macro do processo de Administração de Componentes (KRONIG, COSTA e SPÍNOLA, 2005)

A Administração de Componentes é responsável pelo trabalho metodológico e conceitual dos componentes. As principais atividades são destacadas:

- Definir responsabilidades sobre os componentes: construção, manutenção e consulta;
- Definir critérios de segurança, proteção, integridade, certificação e privacidade dos componentes;
- Desenvolver e implementar políticas, procedimentos e padrões para a construção, manutenção, classificação, catalogação e certificação dos componentes;
- Orientar a confecção de componentes através de padrões;
- Homologar a implementação de novos componentes;
- Estabelecer o controle centralizado dos componentes através da: Biblioteca de Componentes.

A Figura 19 apresenta uma visão-macro do ambiente de desenvolvimento baseado em componentes, integrando os processos de análise de negócio, do desenvolvimento de software e da Administração de Componentes.

O Repositório de Componentes ou Biblioteca de Componentes é a principal ferramenta de apoio e integração, pois permite a busca e recuperação das informações relacionadas aos componentes catalogados.

Como forma de consolidar os conceitos propostos pelos autores KRONIG, COSTA e SPÍNOLA (2005) e pelo RUP, é proposto neste trabalho de dissertação à elaboração de um modelo conceitual como base para projetar e desenvolver uma Biblioteca de Componentes que suporte e automatize o ambiente de desenvolvimento de software a partir de componentes reutilizáveis.

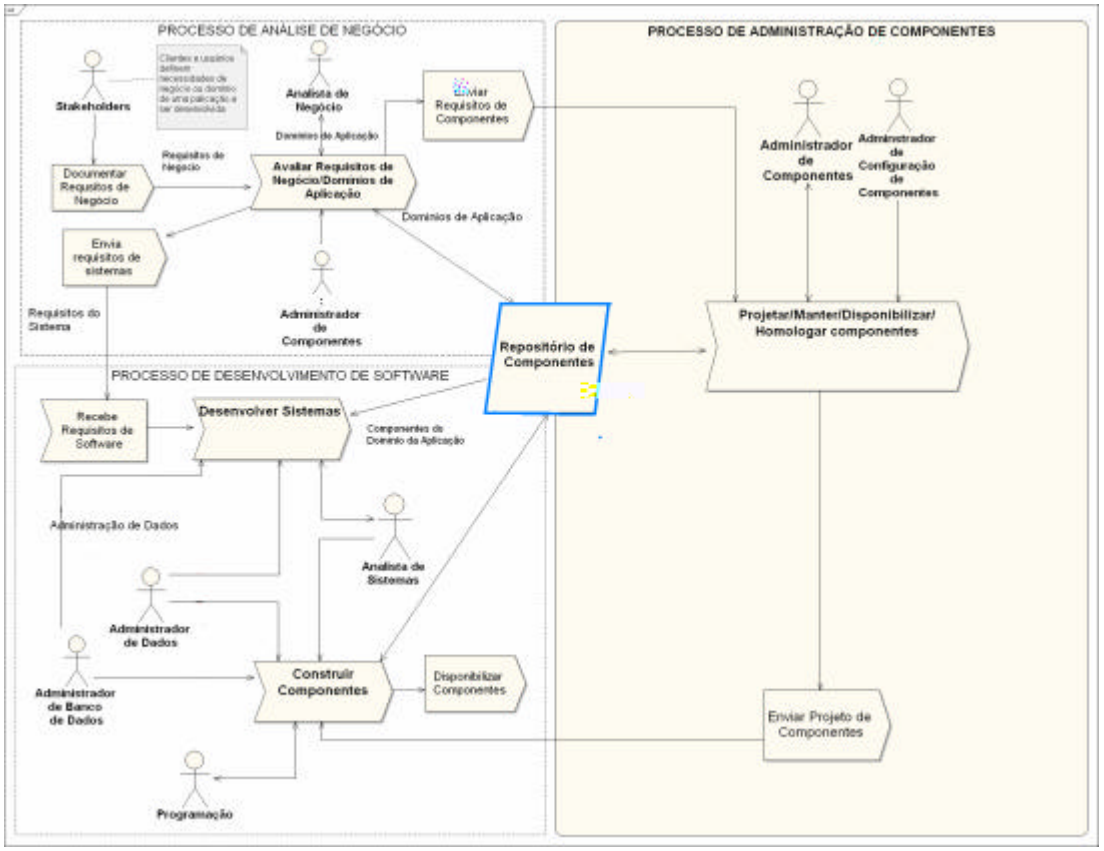


Figura 19 - Visão macro do ambiente de desenvolvimento de software baseado em componentes (KRONIG, COSTA e SPÍNOLA, 2005)

SOMMERVILLE (2003) aponta os seguintes benefícios esperados no de desenvolvimento de software baseado em componentes:

- Maior confiabilidade: os componentes reutilizados que são empregados nos sistemas em operação devem ser mais confiáveis do que os componentes novos, pois eles já foram testados e experimentados em diferentes ambientes;
- Redução dos riscos do processo: com a utilização de um componente existente, serão menores as incertezas sobre os custos relacionados ao reuso desse componente sobre os custos de desenvolvimento, reduzindo assim as incertezas de custos de projeto;
- Conformidade com padrões: alguns padrões, assim como os de interface com o usuário, podem ser implementados como um conjunto de componentes padrões. O uso de interfaces-padrão melhora a confiabilidade, pois os usuários possivelmente comentem menos enganos utilizando uma interface familiar;
- Desenvolvimento acelerado: o reuso de componentes acelera a produção, pois o tempo de desenvolvimento e de validação é reduzido.

5 APOIO AUTOMATIZADO AO REUSO

O desenvolvimento baseado em componentes, conforme pesquisa apresentada necessita de uma infra-estrutura para apoiar as atividades de classificação, busca, armazenamento, divulgação, compartilhamento e entendimento de componentes para a construção de aplicações de software a partir destes componentes.

Para PRESSMAN (2006), uma relação de recursos deve fazer parte da infra-estrutura de apoio à reutilização de componentes de software:

- Banco de dados de componentes capaz de armazenar os componentes de software e a informação necessária para recuperá-los;
- Sistema de gestão de biblioteca que forneça acesso ao banco de dados;
- Sistema de recuperação de componentes de software que possibilite uma aplicação cliente recuperar os componentes e os serviços a partir do servidor de biblioteca; e
- Ferramentas de engenharia de software baseada em componentes que forneçam suporte à integração de componentes em um novo projeto ou implementação.

Cada um destes recursos interage com, ou é incorporado aos limites de uma biblioteca de reuso.

5.1 Biblioteca de Reuso de Componentes

Para CARMA MCCLURE (1997) *apud* ROSSI (2004), a biblioteca de reutilização fornece mecanismos para gerenciar os componentes e para torná-los disponíveis aos projetistas de sistemas de software de uma organização. Tem como funções principais:

- Organizar os componentes de software;
- Armazenar os componentes de software;
- Gerenciar os componentes de software;
- Permitir a existência de múltiplas versões dos componentes de software.

A biblioteca é um elemento de um repositório maior e fornece facilidades para armazenamento de componentes de software. É composta de um banco de dados e de ferramentas para consulta e recuperação destes componentes. Um esquema de classificação de componentes serve de base para as consultas à biblioteca (PRESSMAN, 2006).

No contexto deste trabalho, a biblioteca de componentes é composta de um repositório e de um conjunto de recursos para gerenciar as informações sobre o conhecimento de um componente de software.

O repositório da biblioteca de componentes é uma base de dados projetada para armazenar de modo organizado as informações sobre o conhecimento de um componente de software. O papel do repositório é centralizar e disponibilizar informações que descrevam as características e funcionalidades dos componentes, promovendo o entendimento de como utilizá-los.

O conceito adotado neste trabalho para um componente de software é "um conjunto de código de software (fonte, binário ou executável) que é utilizado sem modificações (BRAUDE, 2005) e deve disponibilizar e executar serviços através de interfaces conhecidas e definidas e possuir a documentação necessária com as informações que auxilie no entendimento de sua funcionalidade".

As informações de um componente foram organizadas e estruturadas a partir das definições propostas por CHEESMAN & DANIELS, (2001) para registrar o conhecimento de um componente:

- Especificação do componente: descrição de uma unidade de software contendo seus comportamentos e estruturas. Os comportamentos são explicitados por meio de suas interfaces;
- Interface do componente: definição de um conjunto de comportamentos que podem ser oferecidos aos seus clientes (outros softwares, aplicações ou sistemas);

- Implementação do componente: realização de uma especificação do componente para que ele possa ser instalado e substituído independentemente de outros componentes;
- Componente instalado: instalação em ambiente de execução (*deployment*) de um componente implementado; e
- Objeto componente: é uma instância de um componente instalado em um ambiente de execução. Um componente instalado pode ter múltiplos objetos componentes.

Para permitir a busca e recuperação das informações relacionadas aos componentes, a solução adotada foi estruturada a partir do método de classificação por facetas (PRIETO-DÍAZ, 1991), apresentado no capítulo 3, seção 3.4. Este método tem sido referenciado e utilizado como base de outras propostas de classificação de informação encontradas na literatura (PFLEEGER, 2004; CASANOVA *et al.,* 2003; VITHARANA *et al.,* 2003; SAMETINGER, 1997).

De acordo com PFLEEGER (2004), o sistema de classificação é amparado por um sistema de recuperação ou repositório, uma biblioteca automatizada, com informações descritivas sobre o componente, que pode procurar e recuperar as informações de acordo com a descrição do usuário.

O sistema de recuperação pode apoiar o administrador da biblioteca a identificar que um tipo específico de componente precisa ser desenvolvido e adicionado à biblioteca.

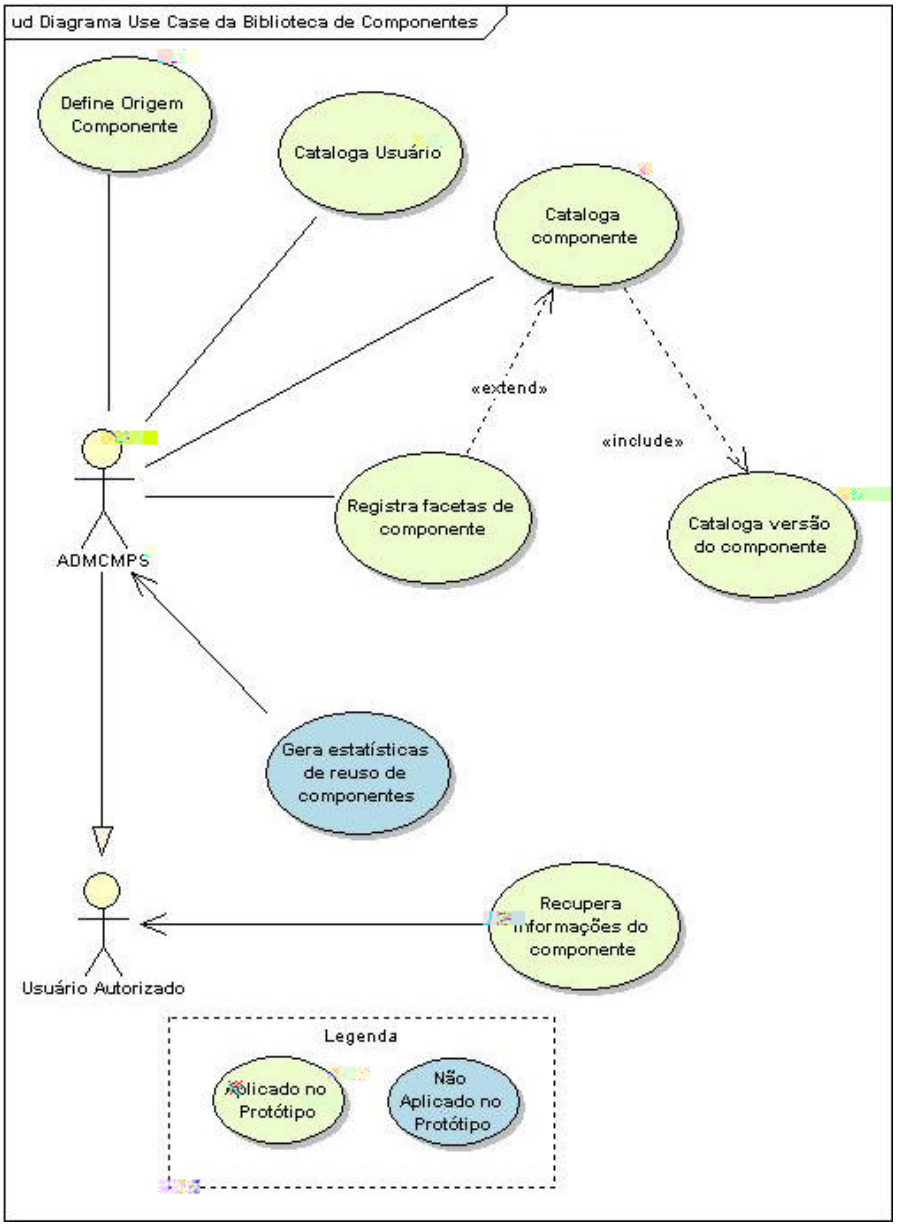


Figura 20 – Visão dos cenários e atores envolvidos na gestão da Biblioteca

elaborada pela autora

O administrador do repositório ou administrador de componentes é responsável pela gerência do conjunto de informação e artefatos relacionados ao componente. Entre as principais atividades:

- Criação e manutenção da estrutura do esquema de classificação dos componentes de software;
- Manutenção das informações relacionadas ao componente de software;
- Atualização das informações sobre as versões do componente;
- Divulgação, para os projetistas, das informações sobre novos componentes; atualizações de versões ou sobre possíveis erros encontrados em componentes; e
- Manutenção dos usuários autorizados (projetistas, desenvolvedores e interessados em utilizar a Biblioteca de Componentes).

Como forma de garantir a qualidade dos componentes armazenados no repositório, também é função do administrador do repositório verificar se os componentes de software candidatos passaram por uma validação.

Para suportar as atividades do administrador de componentes, os principais cenários são implementados:

- Define Origem do Componente: mantém as informações sobre os proprietários e usuários do componente;

- Cataloga Usuário: administra o controle de acesso dos usuários autorizados;
- Cataloga Componentes: disponibiliza recursos para a manutenção das informações dos componentes de software e dos artefatos relacionados. Durante a catalogação, o componente de software deve ser classificado conforme o cenário "Registra Facetas de Componente". Após sua classificação, é armazenado o conjunto de informações para descrever sua funcionalidade, seus comportamentos, questões relativas a sua implementação, como utilizá-lo e aspectos que auxiliam no seu entendimento;
- Cataloga Versão: permite o cadastramento das informações das diferentes versões de um mesmo componente de software;
- Registra Facetas de Componente: recursos para a manutenção da estrutura do esquema de classificação por facetas e recursos para associar as facetas ao componente. O esquema de classificação estabelecido, facilita a busca e a recuperação do componente de software; e
- Gera Estatística de Reutilização: disponibiliza os recursos para permitir a análise de estatísticas sobre a reutilização dos componentes de software armazenados no repositório (não implementada no protótipo).

O cenário “Recupera Informação do Componente” é implementado para disponibilizar aos usuários autorizados à pesquisa de informações relacionadas a um componente de software. Esta pesquisa é realizada através do esquema de classificação por facetas. Esta classificação consiste na seleção de termos das facetas para recuperar as informações dos componentes que satisfaçam esta seleção. Assim, as facetas fornecem as listas pré-definidas de termos que são apresentadas aos usuários para prosseguir na formulação da pesquisa.

Para a aplicação do esquema de classificação, proposto por PRIETO-DÍAZ (1991), é apresentado no quadro 5, as perspectivas ou facetas contempladas no sistema de Biblioteca e uma lista preliminar de termos associados a cada faceta.

c O Quadro 5 - c I R P

- **Faceta Tipo:** caracteriza o tipo de componente, que pode ser de infra-estrutura ou de negócios. Um componente de infra-estrutura fornece serviços independes do negócio da aplicação, sendo suporte aos outros tipos de componentes. Seus serviços podem ser utilizados por vários tipos de componentes e em

diferentes aplicações. Um componente de intra-estrutura é responsável por serviços especializados e realiza funções auxiliares que são compartilhadas e concorridas por vários componentes, como tratamento de mensagens, recursos de apresentação de telas, auditoria e segurança. Disponibiliza informações importantes para a localização de falhas, identificação de uso incorreto do sistema e até de uso não autorizado (SIMÃO, 2002). Um componente de negócio é responsável pela lógica da aplicação, isto é, implementam as regras de negócio do sistema.

- **Faceta Domínio:** estabelece a área de negócio para uma determinada aplicação, como por exemplo, financeiro, comercial, industrial e educação. Um componente é identificado e caracterizado para ser utilizado no desenvolvimento de uma ou mais aplicações em um dado domínio.

- **Faceta Intenção**: refere-se ao aspecto de alterabilidade do componente de reuso e pode ser caixa-branca ou caixa-preta. Na reutilização caixa-branca o componente reutilizado é modificado e adaptado. Na reutilização caixa-preta, o componente de software é reutilizado não é modificado.

- **Faceta Linguagem**: define a linguagem na qual componente foi construído e implementado.

- **Faceta Tecnologia**: especifica o ambiente tecnológico no qual o componente pode ser utilizado, tais como sistema operacional, linguagem de programação, entre outros.
- **Faceta Fabricante:** caracteriza o componente pela sua origem de fornecimento, que pode ser: desenvolvimento interno, fornecedor externo ou componente de prateleira;
- **Faceta Ano Fabricação**: corresponde ao ano de fabricação do componente. Quando não existir esta informação, considera-se o ano de cadastramento do componente na Biblioteca.

A elaboração do critério de busca à Biblioteca é baseada nas sete facetas apresentadas anteriormente. O usuário pode formular a busca selecionando os termos de cada faceta.

A Figura 21 apresenta o diagrama de classes para atender a catalogação, a busca e a recuperação de um componente pelo método de classificação por facetas.

Um diagrama de classe é amplamente usado para descrever os objetos no sistema e os vários tipos de relacionamentos estático que existem entre eles. Também mostra os atributos e operações de uma classe e as restrições à maneira com que os objetos são conectados (FOWLER & SCOTT, 2000).

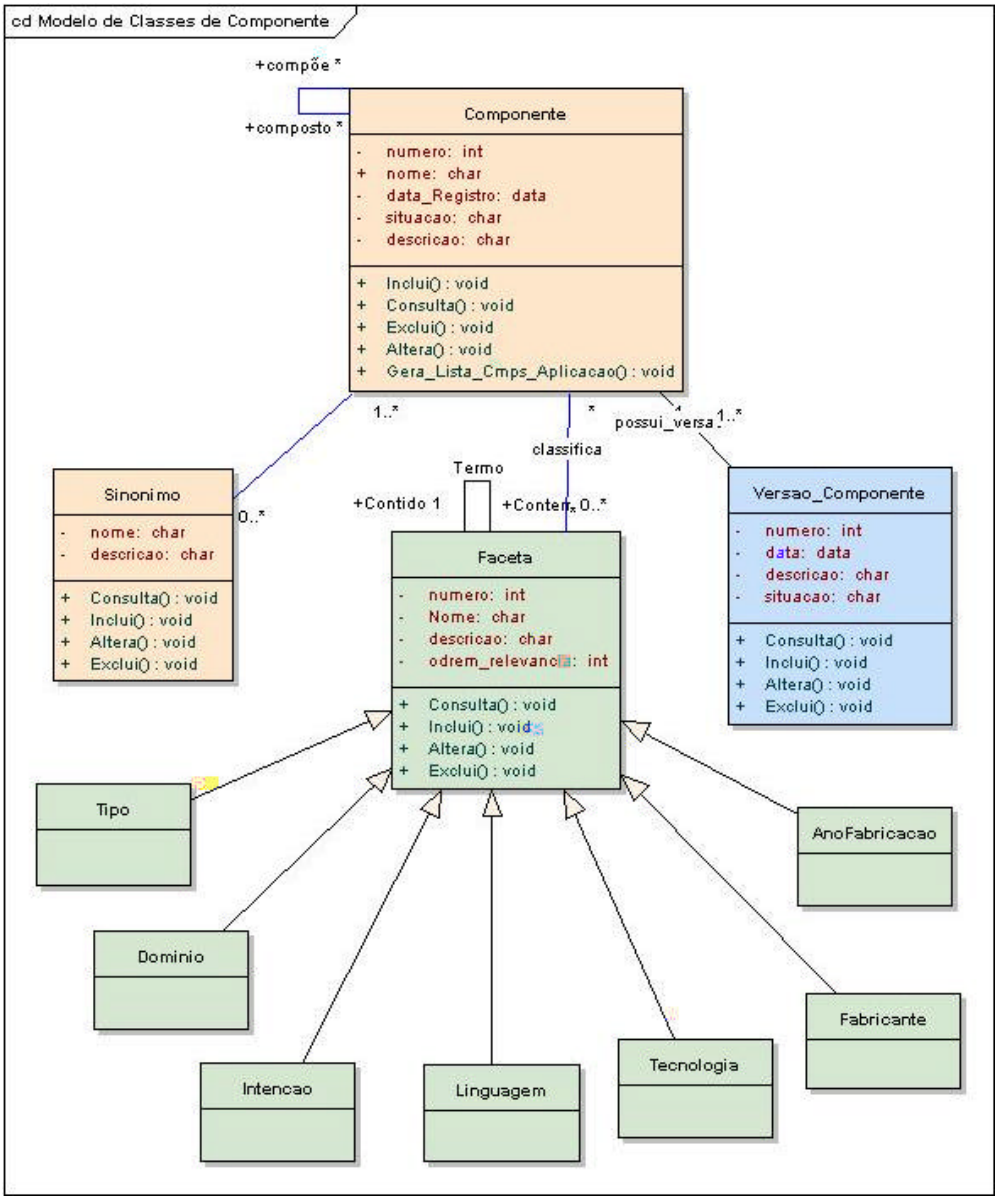


Figura 21 – Diagrama de Classes do Esquema de Classificação

elaborado pela autora

A classe **Componente** identifica o componente a ser armazenado e recuperado e, descreve de modo resumido sua funcionalidade e finalidade:

- numero: identifica unicamente o componente de software;
- nome: especifica o nome do componente;
- data_registro: especifica a data em que o componente de software foi inserido na Biblioteca;
- situacao: especifica a situação do componente: **A** - Componente Ativo; **I** - Componente Inativo (componente usado em outras aplicações e não mais disponível para utilização) ; **C** – Cancelado;
- descricao: descreve os objetivos gerais do componente.

A classe **Faceta** classifica um componente visando sua pesquisa e recuperação. As subclasses **Tipo**, **Domínio**, **Intenção**, **Linguagem**, **Tecnologia**, **Fabricante** e **Ano Fabricação** identificam os tipos de facetas de reuso de um componente, apresentados no Quadro 5. As facetas são ordenadas de acordo com o seu grau de importância.

A classe **Versao_Componente** contém as informações que definem as versões de um componente ao longo do tempo de seu ciclo de vida.

A Classe **Sinônimo** define o conjunto de termos a serem relacionados com um componente de mesmo significado.

Um outro aspecto a ser considerado, mas que não é foco deste trabalho, é o controle sobre as mudanças do componente durante o desenvolvimento e as possíveis manutenções.

Uma gerência de configuração de software ou *Software Configuration Management* (SCM) deve garantir a integridade do componente ao longo de seu ciclo de vida, identificando, organizando, rastreando e controlando as suas modificações.

A Figura 22 apresenta o esquema de navegação para as atividades de gestão da biblioteca.

Para efeito de melhor ilustração, no Anexo A são apresentadas as principais telas de navegação do Sistema de Biblioteca.

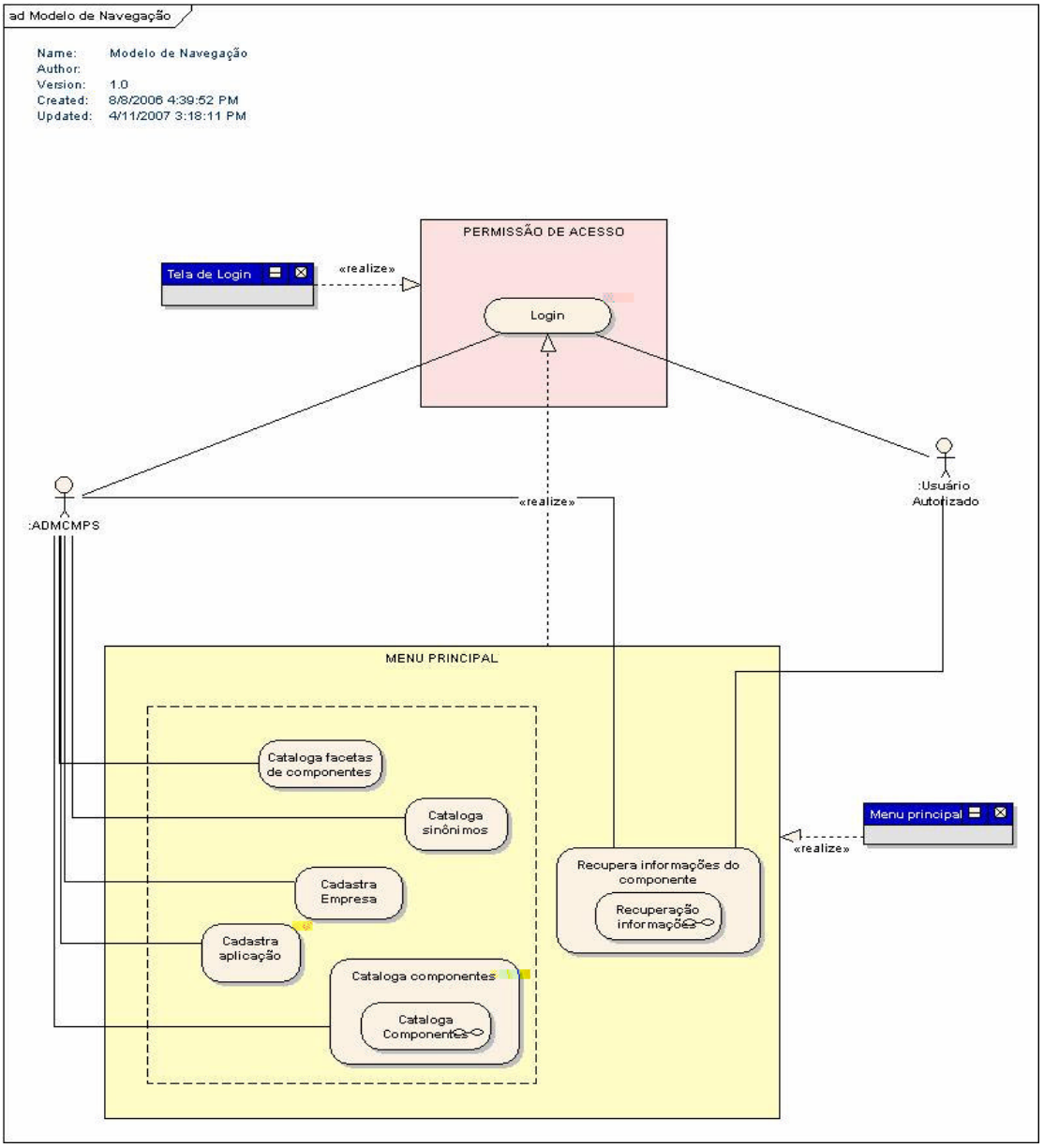


Figura 22 - Esquema de Navegação da Biblioteca de Componentes

6 CONCLUSÃO

Este capítulo sintetiza o trabalho realizado e relaciona suas contribuições para a Engenharia de Software, bem como suas limitações. Aponta para o desenvolvimento de trabalhos futuros como forma de continuidade deste trabalho que podem gerar outras contribuições.

O desenvolvimento de software baseado no reuso de componentes tem como premissa aumentar a produtividade e a qualidade durante o processo de desenvolvimento. A produtividade pode ser aumentada não somente reduzindo o tempo de codificação, mas também os tempos necessários para os testes e documentações.

Para maximizar o reuso de software deve ser implementado um programa sistemático de reuso, com componentes previamente catalogados e documentados em um único local.

De acordo com a literatura pesquisada, uma biblioteca de componentes é a base para apoiar um ambiente formal de reutilização de software, pois toda informação sobre os componentes fica armazenada, padronizada e centralizada.

Por meio da pesquisa realizada foram levantados os conceitos técnicos e científicos para a elaboração de uma base de conhecimento para estabelecer um conjunto de informações para representar e documentar um componente de software reutilizável.

Outros trabalhos e artigos científicos, relacionados com o desenvolvimento e construção de um repositório para armazenar componentes de software, foram encontrados e analisados. Estes apresentam um modelo de um repositório, ou uma biblioteca de componentes, com esquemas de armazenamento e classificação similares a este trabalho. Outros, apresentam apenas um modelo, denominado modelo de referência, para documentar um componente de software. Entretanto, estes modelos são desenvolvidos e direcionados para um ambiente de desenvolvimento de componentes em particular.

Este trabalho, como contribuição à Engenharia de Software, propõe a o desenvolvimento de um modelo conceitual para representar o conhecimento sobre componentes, formando uma base de componentes para ser utilizada em quaisquer ambientes de desenvolvimento de software em fase de implantação ou que já utilizam um programa sistemático de reuso.

Para consolidar este modelo como um produto de software foi desenvolvido um protótipo no Laboratório de Pesquisa de Software do programa de Mestrado em Engenharia de Produção da UNIP – Universidade Paulista.

As informações representadas sobre o conhecimento de um componente propõem ser um ponto de partida para uma pesquisa mais abrangente, não encerrando as definições sobre o conjunto de informações necessário para auxiliar no entendimento das funcionalidades, propriedades,

características e a forma de integração de um componente de software reutilizável.

Em relação aos trabalhos futuros, vários aspectos relacionados ao modelo e ao protótipo precisam ser aprimorados e validados para que a Biblioteca de Componentes apresentada neste trabalho torne-se operacional:

- O protótipo deve ser desenvolvido em uma tecnologia moderna e de preferência aberta (*OpenSource*) aderente à comunidade brasileira desenvolvedora de software;
- Uma relação de componentes de software deve ser projetada ou adquirida para compor a base de dados da biblioteca;
- A aplicabilidade do sistema deve ser testada e avaliada em uma empresa desenvolvedora de software que tenha um programa de reuso implantado ou em fase de implantação; e
- Mecanismo e indicadores devem ser desenvolvidos para analisar o grau de reuso de um componente armazenado na biblioteca.

Além dos componentes de software, uma variedade de outros artefatos de software pode ser desenvolvida com o propósito de reutilização, e que não foram abordados neste trabalho. Estes artefatos incluem especificações, modelos arquiteturais, documentos, padrões e mesmo tarefas relacionadas a processos.

Estas abordagens devem ser pesquisadas para o entendimento e aprimoramento das tecnologias relacionadas com o reuso de software, o que motiva a continuidade deste trabalho.

Como consideração final é importante destacar que o Brasil necessita de um salto qualitativo e quantitativo na produção de software e que este trabalho procurou contribuir com conhecimentos que apóiam a aquisição de *expertise* técnicas em TI e matérias correlatas.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BOHEM, B.W. A Spiral Model of Software Development and Enhancement. **IEEE Computer**, v.21, n.5, p.61-72, May, 1988.

BRAUDE, Eric. **Projeto de Software: da programação à arquitetura: uma abordagem baseada em Java**. Porto Alegre: Bookman, 2005.

BROWN, A. W.; WALLNAU, K.C. The Current State of CBSE. **IEEE Software**, v.13, n. 9, p. 37-46, September/Octuber, 1998.

CASANOVA, M.; STRAETEN V.,R.; JONCKERS ,V. Supporting Evolution in Component-Based Development using Component Libraries. In: **Proceedings of the Seventh European Conference On Software Maintenance And Reengineering** (CSMR'03), IEEE Computer Society, 2003.

COSTA, I. **Contribuição para o Aumento da Qualidade e Produtividade de uma Fábrica de Software através da Padronização do Processo de Recebimento de Serviços de Construção de Softwares**. 2003. Tese de Doutorado – Escola Politécnica da Universidade de São Paulo, São Paulo, 2003.

CHEESMAN, J; DANIELS, J. **UML Components: a simple process for specifying component-based software.** Addison-Wesley, 2001.

CRNKOVIC, I. *et al.* Specification, implementation, and deployment of components. **Association for Computing Machinery. Communications of the ACM**, v. 45, n. 10, p. 35-40, October, 2002.

FOWLER, M.; SCOTT, K. **UML Essencial***,* Porto Alegre: Bookman, 2000.

FRAKES, W.B; KANG, K. Software Reuse Research: Status and Future. **IEEE Transactions on Software Engineering**, v..31, n. 7, p. 529-536, July 2005.

GAMMA, E.; HELM, R.; JOHNSON, R. VLISSIDES, J. **Design Patterns**: **Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995.

GIL, A.C. **Como Elaborar Projetos de Pesquisa**. São Paulo: Editora Atlas, 2002.

GODINHO FILHO, M. **Paradigmas Eestratégicos de Gestão da Manufatura: configuração, relações com o Planejamento e Controle da produção e estudo exploratório na indústria de calçados.** 2004. Tese de Doutorado – Universidade Federal de São Carlos, São Carlos, 2004.

HOPKINS, J. Component Primer. **Association for Computing Machinery. Communications of the ACM**, v.43, n. 10, p. 27-30, Octuber 2000.

IEEE Guide to the Software Engineering Body of Knowledge A project of the IEEE Computer Society Professional Practices Committee, 2004

IEEE/EIA 12207.0-1996**, Standard for Information Technology – Software Life Cycle Process.** IEEE, March 1998.

IEEE STD 1517-1999(R2004)**, IEEE Standard for Information Technology – Software Life Cycle Process – Reuse Process.** Software Engineering Standards Committee of the IEEE Computer Society, June 1999, Realfirmed, March 2004.

IEEE Std 610.12-1990, **IEEE Standard Glossary of Software Engineering Terminology** - Software Engineering Standards Committee of the IEEE Computer Society, December 1990

ISO/IEC 12207:2002/FDAM 2:2004 **Information Technology – Software Life Cycle Processes - Amendment 2,** 2004

KOBRYN, C. Modeling components and frameworks with UML. **Association for Computing Machinery. Communications of the ACM,** v..43, n.10, p.31-38, October 2000.

KOHAN, S. **QuickLocus: proposta de um método de avaliação de processo de desenvolvimento de software em pequenas organizações.** São Paulo, 2003. Mestrado Profissional em Engenharia da Computação – Instituto de Pesquisas Tecnológicas do Estado de São Paulo. São Paulo, 2003.

KRONIG, R; COSTA, I; SPÍNOLA, M M. Uma proposta de um processo prático para apoiar o reuso de software. In: XXV Encontro Nacional De Engenharia De Produção – ENEGEP, 2005, Porto Alegre. **Anais**... Porto Alegre: ABEPRO, 2005.

McCLURE, C..L**. Software Reuse: A Standards-based Guide.** IEEE Computer Society, 2001

MORISIO, M.; TULLY, C.; EZRAN, M. Diversity in reuse processes **IEEE Software**, v.17, n. 4, p. 56-63, July/August 2000.

NBR ISO/IEC 12207/1997 **Tecnologia da Informação – Processos de Ciclo de Vida de Software**, ABNT, Maio 1997.

NONAKA, I ; TAKEUSHI, N. Criação de Conhecimento na Empresa. 12º ed. Rio de Janeiro: Elsevier, 1997.PETERS, J. F.; PEDRYCZ, W. **Engenharia de Software: Teoria e Prática**. Rio de Janeiro: Ed. Campus, 2001.

PFLEEGER, S. L**. Engenharia de Software: Teoria e Prática**. 2º ed. São Paulo: Prentice Hall, 2004.

PRESSMAN, R.S. **Engenharia de Software**. 6º.ed.. São Paulo: McGraw-Hill, 2006.

PRIETO-DIAZ, R., Implementing faceted classification for software reuse. **Communications of the ACM**, v.34, n. 5, p. 88-97, May 1991.

PRIETO-DIAZ, R. Status Report: Software Reusability **IEEE Software**, v.10, n.3, p. 61-66, May 1993.

ROSSI, A.C. **Representação do Componente de software na FARCSoft: ferramenta de apoio à reutilização de componentes de software**. São Paulo 2004. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo. São Paulo, 2004.

ROYCE, W. **Software Project Management: A Unified Framework**. Addison-Wesley, 1998.

RUP. **Rational Unified Process** – Rational Software Corporation. Versão 2002.05.00, 2001.

SAMETINGER, J **Software Engineering with Reusable Components**. Springer, 1997.

SELBY, R.W. Enabling Reuse-Based Software Development of Large-Scale Systems. **IEEE Transactions on Software Engineering**, v..31, n. 6, p. 495-510 , June 2005.

SIMÃO, R. **A. Características de Qualidade para Componentes de Software.** 2002**.** 162 f. Dissertação (Mestrado) - Universidade de Fortaleza – UNIFOR. Fortaleza, 2002.

SLACK, N *et al.* **Administração da Produção**. São Paulo. Editora Atlas, 1997.

SOMMERVILLE, I**. Engenharia de Software**. 6º ed. São Paulo: Addison Wesley, 2003.

SWEBOK. **IEEE Guide to the Software Engineering Body of Knowledge.** A project of the IEEE Computer Society Professional Practices Committee, 2004

TRINDADE, A.L.P. **Uma Contribuição para o Entendimento do Papel da Usinagem na Preservação do Conhecimento em Ambientes de Fábrica Fábrica de Software**. São Paulo 2006. Tese de Doutorado - Escola Politécnica da Universidade de São Paulo. São Paulo, 2006.

TURBAN, E. *et al*. **Administração de Tecnologia da Informação**. Rio de Janeiro, Elsevier, 2005.

VITHARANA, P; ZAHEDI, F.M.; JAIN, H. Knowledge-based repository scheme for storing and retrieving business components: a theoretical design and an empirical analysis. **IEEE Transactions on Software Engineering**, v..29, n. 7, p. 649-664, July 2003.

YOURDON, E.; ARGILA, C. **Análise e Projeto Orientados a Objetos**. São Paulo: Makron Books, 1999.

GLOSSÁRIO

Arquitetura de software: uma estrutura global de um sistema de software com seus principais subsistemas, incluindo a especificação destes subsistemas, seus inter-relacionamentos e a tecnologia apropriada (SAMETINGER, 1997; CHEESMAN & DANIELS, 2001).

Ativo (*asset*): Um item, como especificações, código-fonte, documentação, componentes, plano de testes, procedimentos manuais, entre outros que foram projetados para uso em múltiplos contextos. Este termo foi redefinido a partir dos padrões do IEEE, para adequar seu significado mais corretamente no contexto de reuso de software (IEEE-Std 1517 – 1999(R2004), 2004).

Domínio *(domain): "problem space",* isto é, representa um segmento da organização na qual existe uma forte possibilidade para aplicar o reuso (IEEE-Std 1517 – 1999(R2004), 2004; McClure, 2001).

Engenharia de domínio: É uma abordagem baseada no reuso para definir o escopo (definição do domínio), especificar a estrutura (arquitetura do domínio) e construir os ativos (requisitos, código-fonte, componentes, dcoumentação) para uma classe de sistemas, subsistemas ou aplicações. O objetivo da engenharia de domínio é identificar, construir, catalogar e disseminar um conjunto de componentes de software em um domínio de aplicação particular (IEEE-Std 1517 – 1999(R2004), 2004; McClure, 2001).

ANEXO A : PRINCIPAIS TELAS DO SISTEMA DE BIBLIOTECA

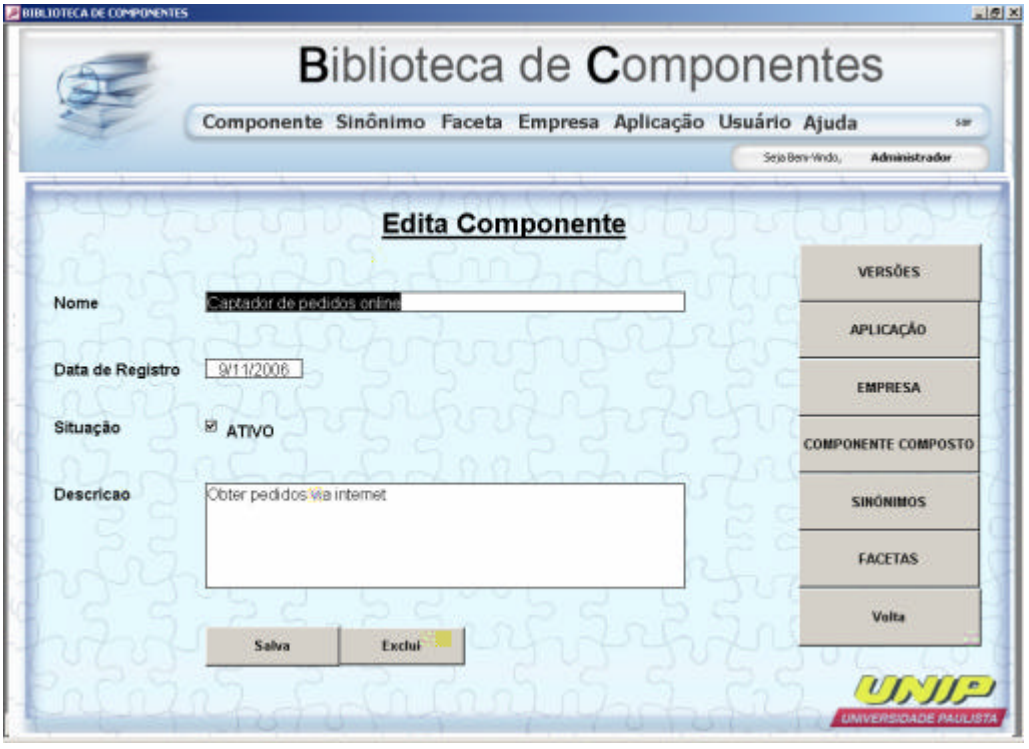
(Administrador de Componentes)

Tela de Login: Permissão de Acesso

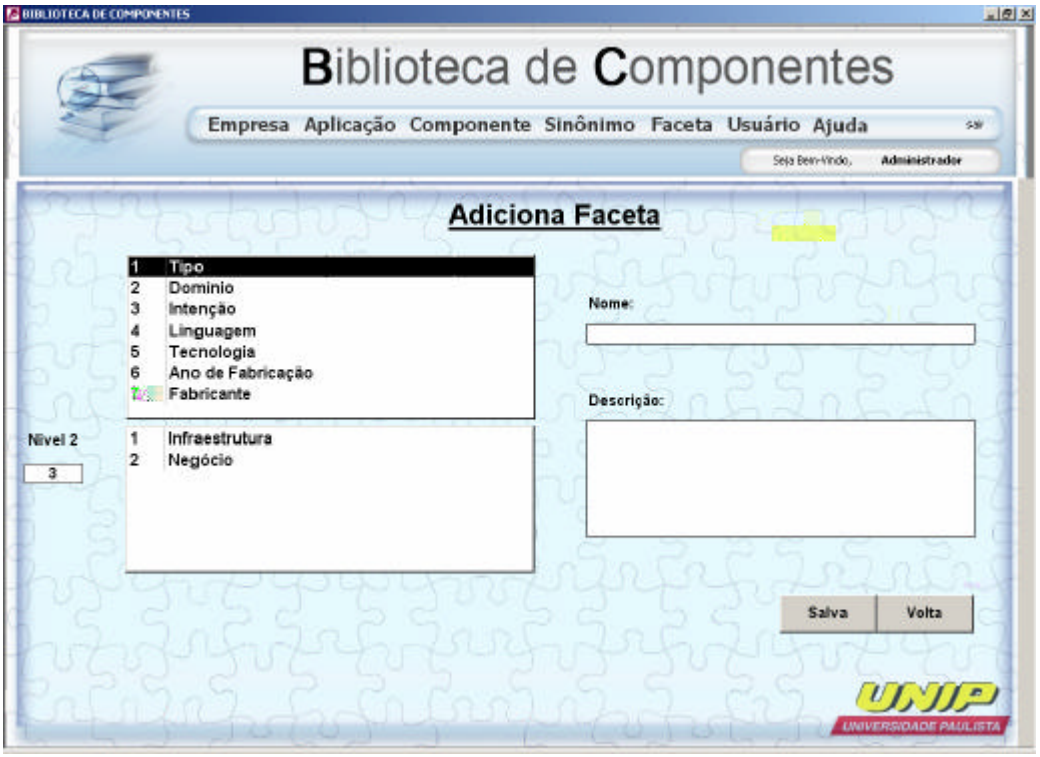
Menu Principal

Cataloga Componente

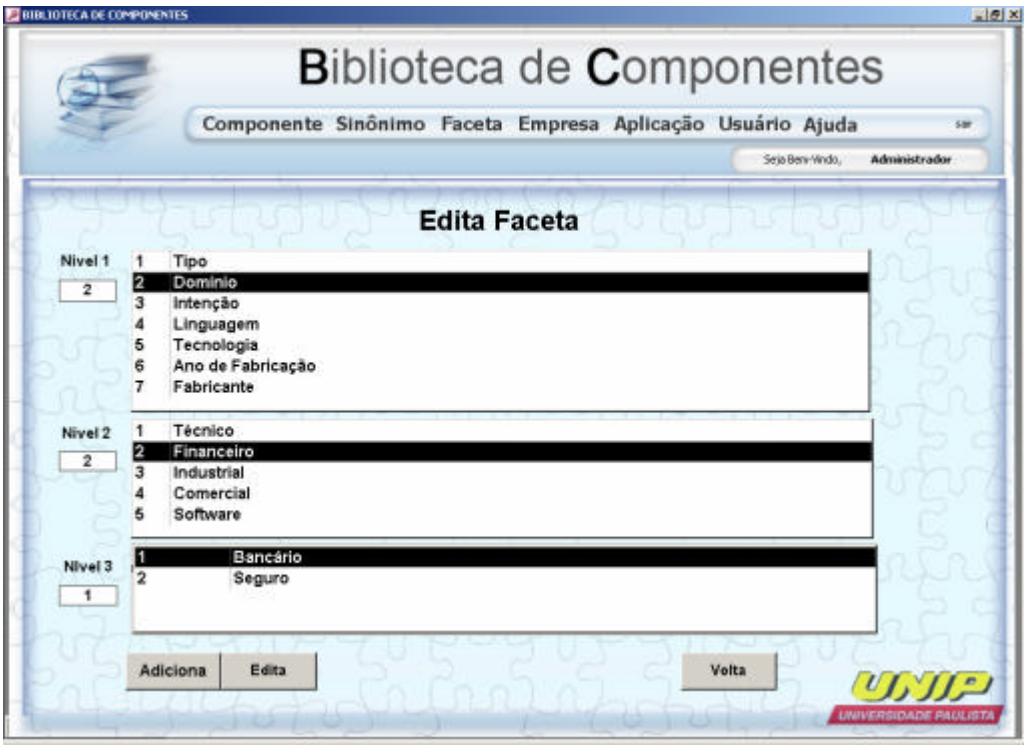
Consulta Componente



Cataloga Facetas para Classificar e Recuperar Componentes



Consulta Faceta: Domínio



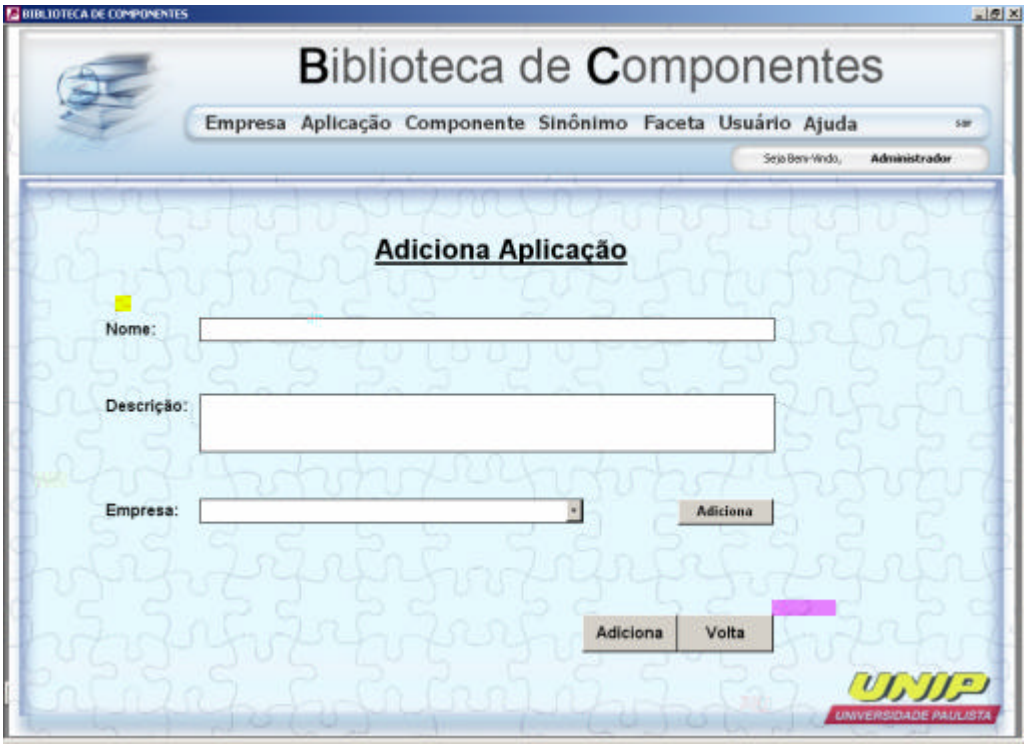
Associa Facetas ao Componente

Consulta Facetas Associadas ao Componente

UNIVERSIDADE PAULISTA - UNIP
Programa de Pós-Graduação em Engenharia de Produção (Mestrado)

Proposta de uma base de conhecimento sobre componentes para apoiar o desenvolvimento de software

Cataloga Aplicação



Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo