GABRIEL LUNA FREIRE RESENDE

UMA ANÁLISE COMPARATIVA DE SISTEMAS DE GESTÃO DE COMPONENTES DE SOFTWARE

LAVRAS - MG
2010

GABRIEL LUNA FREIRE RESENDE

UMA ANÁLISE COMPARATIVA ENTRE SISTEMAS DE GESTÃO DE COMPONENTES DE SOFTWARE

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

Orientador
Dr. Antônio Maria Pereira de Resende

LAVRAS - MG
2010

GABRIEL LUNA FREIRE RESENDE

ANÁLISE COMPARATIVA ENTRE SISTEMAS DE GESTÃO DE COMPONENTES DE SOFTWARE

Monografia apresentada ao Colegiado do Curso de Ciência da Computação, para a obtenção do título de Bacharel.

APROVADA em 17 de Junho de 2010

Prof. Dr. André Luiz Zambalde	UFLA
Prof. Dr. Heitor Augustus Xavier Costa	UFLA

Prof. Dr. Antônio Maria Pereira de Resende
(Orientador)

LAVRAS – MG
2010

AGRADECIMENTOS

Agradeço principalmente a Deus, se eu cheguei até aqui foi por que Ele me guiou e me protegeu.

Aos meus pais, pelo amor, incentivo nesses anos todos. Agradeço ao meu irmão pela cumplicidade, amizade e apoio. Agradeço a minha namorada pelo companheirismo e pelo amor durante estes anos todos.

Agradeço ao meu orientador, Antônio Maria pela paciência e disposição para me orientar. Obrigado pelas conversas, correções, sugestões e dicas.

Um obrigado todo especial aos meus amigos de graduação, que sempre me ajudou a superar os momentos mais difíceis nesta jornada.

Aos professores do DCC pela dedicação, profissionalismo e entusiasmo durante todo o curso.

Aos funcionários do DCC pelos favores e pelos problemas resolvidos com toda atenção e simpatia durante o curso.

RESUMO

A técnica de reúso de software contribui com os principais objetivos da Engenharia de Software como o aumento da qualidade, da produtividade e da eficiência no desenvolvimento de software. Porém, essa técnica possui atividades como a catalogação, armazenamento, busca e recuperação de componentes que dificultam sua aplicação. Para amenizar essas dificuldades, a Engenharia de Software apóia-se em ferramentas de Sistemas de Gestão de Componentes (SGC) que auxiliam na catalogação, armazenamento, busca, recuperação e gerenciamento de componentes de software. Além disto, o sucesso do Desenvolvimento Baseado em Componentes (DBC) depende, diretamente, da qualidade e aplicação do SGC. Devido a variedade de SGCs existentes no mercado e sua importância no sucesso da aplicação do reúso, apresenta-se, neste trabalho, uma análise comparativa entre os principais SGCs, objetivando-se gerar um quadro comparativo capaz de auxiliar os Engenheiros de Software a escolher o melhor SGC para seu projeto.

Palavras-chave: Engenharia de Software. Reuso. Sistema de Gestão de Componentes. Software.

ABSTRACT

The technique of reusing software helps with the main goals of Software Engineering as increased quality, productivity and efficiency in software development. However, this technique has activities such as cataloging, storage, search and recovery of components that hinder its application. To alleviate these difficulties, the Software Engineering relies on tools for Component Management Systems (CMS) that assist in cataloging, storage, retrieval and management software components. Moreover, the success of Component Based Development (CBD) depends directly on the quality and implementation of the GSC. Given the variety of SGCs in the market and its importance in the successful implementation of reuse, we present in this work, a comparative analysis between the main SGCs, aiming to generate a comparison table can help software engineers to choose GSC best for your project.

Key-words: Software Engineering. Reuse. Management System Components. Software.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Com o crescimento na área de informática e o Desenvolvimento de Software Baseado em Componentes, novas ferramentas foram criadas para auxiliar e aperfeiçoar a produção de software.

Uma das estratégias adotadas pela Engenharia de Software foi a utilização do reúso que ajuda a solucionar possíveis dificuldades ao decorrer da produção. Kontio;Caldeira; Basili (1996) acreditam que esta estratégia contribui para qualidade e produtividade no desenvolvimento do software.

O Desenvolvimento Baseado em Componentes (DBC) consiste em uma montagem de software utilizando componentes que já estão prontos e que possuem certa "maturidade" para que sua aceitação seja a melhor possível. Com o DBC, a criação de software resume-se em adaptar componentes por meio de interfaces bem definidas, o que é um aspecto chave para o reúso e independência entre os módulos do sistema.

Buscando o desenvolvimento com menor esforço, algumas empresas estão adotando o DBC com a finalidade de aumentar a produtividade, entretanto, o reúso de componentes apresenta alguns fatores limitantes. Bass; Buhman; Dorda; Long; Robert; Seacord; Wallnau (2000) identifica a carência de componentes, padrões, certificação e métodos para a construção de componentes como os principais fatores inibidores para o uso de componentes. Para Crnkovic (2003), destaca-se a atual dificuldade em identificar, selecionar, negociar e recuperar componentes que atendam aos requisitos especificados, e, além disso, que possuam alto grau de reusabilidade e qualidade.

Segundo Frakes; Kang (2005), além de prover facilidades para armazenamento de componentes, os Sistemas de Gestão de Componentes devem disponibilizar interfaces para busca de componentes, adotar metadados para representação dos componentes, prover mecanismos para certificação de

qualidade, incluir facilidades de controle de versão e métricas de reúso, como também, adotar mecanismos que viabilizem o retorno de investimento na negociação de componentes.

Assim, para viabilizar as abordagens do DBC, a adoção de um Sistema de Gestão de Componentes é, além de uma característica desejável, uma necessidade, pois segundo Guo (2005), o Sistema de Gestão de Componentes tem o propósito de auxiliar no reúso de componentes, e, assim, melhorar as metas de qualidade, custo e produtividade. Vale ressaltar que, segundo Searcord (1999), o Sistema de Gestão de Componentes atua como um elo entre o desenvolvimento para reúso e o desenvolvimento com reúso de componentes, provendo meios para armazenamento, busca e recuperação dos mesmos. Além disso, devem-se considerar ainda diversos fatores importantes como a qualidade, o suporte de fornecedores, os custos, as cláusulas contratuais e a conformidade entre componentes e requisitos. Com isto, muitas organizações perdem tempo neste processo de seleção.

Segundo Gimenes; Travassos (2002), a engenharia de domínio, os *frameworks*, os padrões, as arquiteturas de software e o Desenvolvimento Baseado em Componentes são algumas abordagens utilizadas na Engenharia de Software que favorecem a reutilização.

Visando favorecer o DBC, criou-se a necessidade de realizar análises comparativas entre os principais Sistemas de Gestão de Componentes de Software com o objetivo de gerar informações que auxilie e facilitem a escolha de um Sistema de Gestão de Componentes.

1.1 Objetivos

Este trabalho objetiva coletar as principais características dos Sistemas de Gestão de Componentes de Software e compará-las, gerando informações que

auxilie e facilitem a escolha de um Sistema de Gestão de Componentes, tanto para uso empresarial quanto pessoal.

1.2 Estrutura do Trabalho

O presente trabalho encontra-se estruturado do seguinte modo:

- No capítulo 2 encontra-se o Referencial Teórico onde serão fundamentados os principais conceitos necessários para este trabalho;
- No capítulo 3 são apresentados os principais Sistemas de Gestão de Componentes, entre eles estão***: Agora***, ***Spars-J***, ***Sensedia***, ***ComponentSource*** e ***CodeBroker****;*
- No capítulo 4 está definida a metodologia utilizada, contendo os métodos da pesquisa;
- No capítulo 5 é apresentada a analise comparativa realizada entre os Sistemas de Gestão de Componentes apresentados no capítulo 3 e os resultados obtidos;
- O capítulo 6 é apresentado às conclusões, contribuições e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo, são descritas as informações básicas necessárias ao entendimento da essência desde trabalho.

2.1 Engenharia de Software Baseada em Componentes (ESBC)

A ESBC, superficialmente parece bastante semelhante à engenharia de software convencional ou orientada a objetos. Segundo Peters (2001), o processo começa quando uma equipe de software estabelece os requisitos para um sistema a ser construído usando técnicas de coleta de requisitos convencionais. Um projeto arquitetural é estabelecido, mas ao invés de passar imediatamente a tarefas de projeto mais detalhadas, a equipe examina os requisitos para determinar que subconjunto é mais adequado à composição de componentes, do que à construção. Isto é, a equipe formula as seguintes questões para cada requisito do sistema (Peters, 2001):

- Componentes comerciais prontos para uso (*Commercial Off-The-Shelf (*COTS)) estão disponíveis para implementar o requisito?
- Componentes reusáveis, internamente desenvolvidos, estão disponíveis para implementar o requisito?
- As interfaces dos componentes disponíveis são compatíveis com a arquitetura do sistema a ser construído?

A equipe tenta modificar ou remover os requisitos do sistema que não podem ser implementados com COTS ou com componentes próprios. Se os requisitos não puderem ser mudados ou descartados, os métodos convencionais ou de orientação a objetos de engenharia de software são aplicados para

desenvolver aqueles componentes novos, que precisam ser trabalhados pela engenharia para satisfazer os requisitos.

2.1.1 Conceitos Relativos a Componentes

Segundo Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit (2007), a analogia que pode auxiliar na compreensão sobre componentes de software é o brinquedo Lego®. As peças do clássico brinquedo de montar podem simbolizar a idéia de componentes de software em um sentido mais geral. As peças do Lego® permitem a montagem de automóveis, casas, aviões, navios, em um grande número de possibilidades cujos limites são dados pela variedade de formatos, pela intercambialidade das peças e pela criatividade do "construtor". Algumas peças podem ser empregadas na construção de qualquer objeto. Outras têm funções muito específicas e, portanto, menor intercambialidade.

A analogia é simples, mas é possível extrair dela algumas características básicas de componentes:

- Componentes são intercambiáveis;
- Componentes são reutilizáveis;
- Alguns são de uso mais geral e outros têm uso mais específico; e
- Componentes interagem com outros componentes.

Do ponto de vista da engenharia de software e considerando as características citadas acima, pode-se inicialmente definir um componente de software como um programa de computador com função claramente definida, modularizado, integrável com outros a partir de interfaces igualmente definidas.

A literatura registra diversas definições para componentes de software. Para referência neste trabalho, é adotada a definição de Rossi (2004):

> *"Um componente (em geral) é um pacote coerente de artefatos de software que pode ser independentemente desenvolvido e distribuído como uma unidade, e que pode ser composto, sem alterações, com outros componentes para construir algo maior".*

Segundo Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit (2007), o princípio que levou ao desenvolvimento de componentes foi o de otimizar o desenvolvimento de novos softwares a partir de partes de programas já existentes (reuso de software). Trata-se de um processo natural de padronização de partes, como em tantas outras indústrias, porém em formato intangível. Assim, soluções complexas podem ser obtidas a partir da integração ou composição de diversos componentes, à semelhança de qualquer produto que se monta a partir de suas partes. Naturalmente, o grau de uso de componentes para produzir um software completo deve variar em função dos níveis de desenvolvimento requeridos para cada solução. A componentização é um fenômeno limitado pela própria criatividade e inventividade que se aplica ao desenvolvimento de cada produto.

A tecnologia de componentes de software refere-se às tecnologias relacionadas ao desenvolvimento e uso de componentes de software, ou seja, ferramentas que auxiliam no projeto, construção, combinação, configuração e customização final dos componentes. Assim, há duas principais atividades ligadas a componentes: o desenvolvimento e a produção dos componentes e seu uso para desenvolvimento e produção de programas (Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit, 2007).

A Engenharia de Software Baseada em Componentes (ESBC) envolve as práticas necessárias para o desenvolvimento baseado em componentes de

forma sistemática, define diversas características fundamentais e estuda as vantagens e desvantagens da adoção de componentes.

2.1.2 Classificação de Componentes de Software

Para efeito de classificação dos componentes do ponto de vista de seu uso e especificidade, neste trabalho adotou-se a divisão proposta por Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit (2007):

- **Componentes genéricos**: são componentes de uso comum em muitos sistemas, tais como os componentes de interface com os usuários também conhecidos como *Graphical User Interface* (GUI);
- **Componentes de serviços**: são componentes que fornecem serviços especializados, mas não são específicos do ponto de vista de domínio de aplicação, como componentes para tratamento de erros em comunicação de dados, criptografia, segurança, geração de gráficos, etc.;
- **Componentes de domínio**: são componentes específicos para domínios definidos, que implementam regras (de simples a complexas) de negócios, por exemplo, regras do setor financeiro ou de construção civil.

Nesta perspectiva, a Figura 1 apresenta estas divisões e sua distribuição em termos de intensidade potencial de reúso e de especificidade de aplicação do componente. Um componente genérico tem uma aplicabilidade comum a diversos domínios e, por isso, apresenta maior potencial de reúso. No outro

extremo, estão os componentes com conhecimento de regras de negócio, mais específicos, e que têm menor possibilidade de reúso.

É fundamental destacar, entretanto, que esta é apenas uma das classificações possíveis. Como em boa parte das tentativas de sistematizar e categorizar software, os limites entre as categorias não são claramente definidos e há zonas fronteiriças de difícil definição.

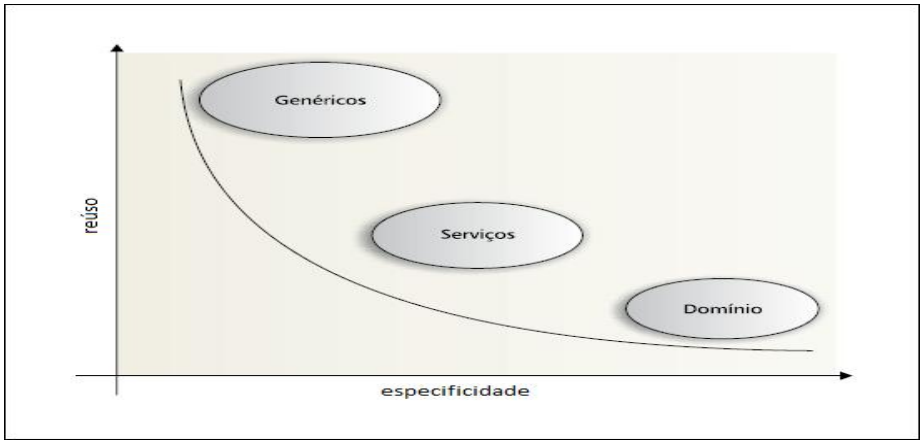

Figura 1 Distribuição de Componentes por Grau de Reúso vs. Especificação de Aplicação
Fonte: Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit (2007)

2.1.3 Reúso e Componentização

Como comentado, a principal motivação para o uso de componentes é a potencialização do reúso de software. Considerando que há escala na produção de componentes, os custos de produção por unidade de produto tendem à redução, com possíveis ganhos de produtividade associados.

Essa questão, entretanto, frequenta os conceitos de engenharia de software há décadas. A ESBC é um passo a mais no reúso de código, exatamente por possibilitar a reutilização de componente “como ele é”, ou seja, o

componente é reutilizado sem alteração de sua implementação, sem custos de desenvolvimento, apenas de "montagem". Segundo Szyperski (2003), para melhor compreensão, pode-se dividir o reúso de software em quatro categorias:

a) **Reúso de código fonte**: trechos de código reutilizável são usados durante a fase de desenvolvimento de um novo software (copiados e colados);
b) **Reúso de partes de software**: reúso de arquitetura e implementação de fragmentos de software em diferentes projetos. Exige um processo de desenvolvimento mais elaborado. O reúso ocorre durante o desenho da arquitetura do projeto e da implementação do código. Assim, não existe componente como uma parte identificável na aplicação final, não sendo substituído com facilidade;
c) **Integração dinâmica de componentes de diversas fontes**: o reúso não ocorre na fase de desenvolvimento do software. A aplicação já está desenvolvida e novas funcionalidades são acrescentadas a partir de softwares *plug-ins*. São exemplos deste tipo de componente os *plug-ins* adicionados aos browsers para que estes consigam visualizar arquivos em formato P-DF;
d) **Componentização**: esta categoria é a mais complexa. É sua característica que a atualização, a extensão do sistema e a integração possam acontecer dinamicamente. Isto permite que os componentes sejam utilizados além das fronteiras das organizações. Nesta categoria, estão concentradas as pesquisas do momento e, também, a revolução potencial que pode ser proporcionada pela tecnologia de componentes.

A engenharia de software tradicional contempla reúso, mas apenas nas três primeiras categorias relacionadas. A ESBC tem como objetivo atingir a última categoria de reúso usando procedimentos, métodos e regras para o desenvolvimento de componentes.

2.1.4 Vantagens e Desvantagens Técnicas do Uso de Componentes

Além dos ganhos de produtividade associados ao reúso, encontram-se ganhos de qualidade e funcionalidade, ambos proporcionados pela tecnologia. Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit (2007) argumenta que o fato de um componente ser reutilizado em diferentes e numerosas situações faz com que ele seja mais testado e tenha seus erros corrigidos, atingindo a maturidade mais rapidamente. Apesar de parecer uma relação causal direta entre qualidade/funcionalidade e a tecnologia de componentes, este fato ainda é controverso na literatura e não há evidências empíricas contundentes. Pode-se dizer que há uma tendência de que isto ocorra caso surja um mercado competitivo de componentes. Outra vantagem técnica atribuída ao uso de componentes é a tecnologia produzir softwares mais flexíveis, duráveis e de manutenção facilitada.

Segundo Alves; Lucca; Carneiro; Ferreira; Veiga; Minoda; Tacca; Rezende; Wit (2007), a maioria das desvantagens da tecnologia de componentes está ligada à sua relativa imaturidade. Esta imaturidade é evidenciada pelo grande número de problemas ainda em aberto na tecnologia e é reflexo principalmente do fato de que nenhum padrão seja predominante, o que gera incertezas com relação a investimentos. Outra decorrência da ausência de um padrão predominante é a dificuldade de manter as vantagens acima referidas.

2.2 Desenvolvimento Baseado em Componentes (DBC)

Segundo D'Sousa; Wills (1999), o DBC é definido em como uma abordagem de criação de sistemas de software em que todos os artefatos (do código fonte até as especificações das interfaces, arquiteturas e modelos de negócios) podem ser construídos, montando-se, adaptando-se e interconectando-se componentes existentes em diversas configurações. Nessa abordagem, alguns componentes devem ser intencionalmente desenvolvidos, enquanto outros, descobertos e adaptados.

O DBC está inovando o modo de construção de software de modo que em um futuro promissor pessoas que não são programadores poderão construir software simplesmente juntando os componentes e utilizando ferramentas de auxílio.

Segundo Kotonya; Hutchinson; Sawyer; Walter; Joan (2002), o DBC possui duas linhas de pesquisa principais: a da Engenharia de Componentes e a da Engenharia da Aplicação.

A Engenharia de Componentes tem por objetivo desenvolver componentes reutilizáveis para a construção de sistemas e suas principais responsabilidades consistem em identificar, construir, catalogar e disseminar componentes de software pertinentes a um domínio específico de aplicação.

A Engenharia da Aplicação tem por objetivo propiciar a aplicação dos componentes de software e suas principais responsabilidades consistem em qualificar, adaptar e integrar componentes para utilização em novos sistemas.

Segundo Brown; Short (1997), há cinco etapas neste processo: Seleção, Qualificação, Adaptação, Composição e Atualização.

A etapa de Seleção de Componentes faz a busca e seleção de componentes que têm potencial para serem usados na construção do sistema. A

busca por componentes envolve tanto COTS quanto componentes que foram usados em outros sistemas desenvolvidos anteriormente.

A etapa de Qualificação do Componente faz análise dos componentes reusáveis, para averiguar se em que proporção se adequam aos requisitos do estilo arquitetural do sistema. Há três importantes aspectos considerados: performance, confiabilidade e usabilidade.

A etapa de Adaptação do Componente faz modificações dos aspectos do componente. É um passo necessário, pois raramente um componente se adaptará de pronto ao sistema. Dependendo do tipo de componente (e.g. COTS ou *in-house* [feito na própria empresa]) diferentes estratégias de adaptação podem ser empregadas.

A etapa de Composição faz a integração dos componentes no sistema, por meio de uma infra-estrutura feita para aglutinar os componentes em um sistema operacional. A infra-estrutura geralmente é em si mesma, uma biblioteca especializada de componentes e serviços que habilita coordenação entre os componentes e realização de tarefas.

A etapa de Atualização faz a atualização dos componentes, onde versões antigas serão substituídas ou novos componentes, com comportamento e interface similares, serão incluídos.

2.3 Sistema de Gestão de Componentes (SGC)

Segundo Rossi (2004), SGCs é uma ferramenta cujo objetivo é apoiar o administrador do repositório em suas principais atividades como gerenciamento de componentes (inserção, modificação, exclusão, notificação de mudanças, etc.) e manutenção de usuários dentro do repositório.

Os Sistemas de Gestão de Componentes (SGC) juntamente com Repositórios de Componentes são uma excelente motivação para alcançar o reúso, pois auxiliam no armazenamento e recuperação de componentes.

Segundo Sametinger (1997), um Repositório de Componente é um local preparado para armazenamento e recuperação de componentes. Considera-se importante o armazenamento de informações adicionais sobre componentes, visando uma recuperação eficiente. Isto é, a probabilidade de componentes serem reutilizados está ligada diretamente à sua disponibilidade dentro do repositório, incluindo a facilidade de localizá-los e recupera-los. Dentre os repositórios de componentes existentes, Sametiger (1997) distingue três tipos:

- **Repositórios locais**: correspondem aos repositórios que armazenam componentes de propósito geral em um repositório único;
- **Repositórios específicos a um domínio**: correspondem aos repositórios que armazenam componentes específicos, com escopo bem definido, podendo prover componentes alternativos para as mesmas tarefas;
- **Repositórios de referência**: correspondem aos repositórios que auxiliam na busca por componentes em outros repositórios.

Para Seacord (1999), Repositórios de Componentes convencionais historicamente falharam por serem concebidos como sistemas locais e centralizados. Dessa forma, possuem acessibilidade e escalabilidade limitada e controle exclusivo dos componentes catalogados. O autor ainda afirma que é necessário adicionar mais semântica aos componentes armazenados no repositório. Autores como Grundy (2000) reforçam esta opinião e afirmam que muitos Repositórios juntamente com os SGCs foram desenvolvidos provavelmente estendendo abordagens usadas por bibliotecas de software,

porém estas abordagens possuem algumas deficiências que incluem consultas baseadas em serviços, falta de um alto nível de descrição das capacidades dos componentes e falta de validação ou verificação dos componentes recuperados.

Constantopoulos; Doerr; Vassiliou (1993) afirma que SGCs construídos com reúso em mente, podem ser considerados sistemas de informação com um propósito especial, exigem um poderoso suporte a modelagem semântica e flexível recuperação de informações. Neste sentido, SGCs permitem armazenar e recuperar componentes durante a análise, o projeto e a implementação de um sistema. Além dos componentes, estes SGCs armazenam descrições de interfaces, propriedades não-funcionais, implementações e casos de testes.

Segundo Sametinger (1997), SGCs são considerados a ligação entre o desenvolvimento com componentes e o desenvolvimento de componente. O autor afirma ainda que a probabilidade de um programador reutilizar um componente em detrimento a sua construção depende, essencialmente, de fatores como a disponibilidade do componente no repositório, o mecanismo de busca e a habilidade do programador em pesquisar no SGC.

Segundo Henninger (1997), SGC pode ser utilizado para diversos propósitos. Entre eles destacam-se:

- **Armazenamento de modelos** – um processo de desenvolvimento envolve a geração de vários artefatos que normalmente são armazenados em locais diferentes. Um SGC juntamente com o Repositório de Componente possibilita a centralização do armazenamento desses artefatos e o controle de cada uma das suas versões;
- **Integração de ferramentas** – muitas vezes os modelos gerados durante um processo de desenvolvimento originam-se de ferramentas distintas. Assim, para a troca de informações entre elas,

é necessário que haja uma “linguagem” para que as mesmas possam se comunicar. O Sistema de Gestão de Componentes pode servir para um armazenamento temporário e para a intermediação entre as ferramentas, ou seja, pode transformar o modelo de entrada de uma delas em um modelo compatível para a outra;

- **Manutenção de sistemas legados** – um Sistema de Gestão de Componente pode ser útil para o armazenamento de dados relacionados a sistemas antigos. Esses dados podem posteriormente ser analisados para determinar os possíveis impactos gerados por modificações.

Sistema de Gestão de Componentes possui duas funcionalidades consideradas principais, a pesquisa e a recuperação dos componentes. No entanto, a presença de algumas outras funcionalidades pode facilitar e, com isso, incentivar a sua utilização.

Conforme proposto por Ezran (1999), serão apresentadas, a seguir, algumas das principais funções que um repositório pode disponibilizar aos seus usuários. Vale ressaltar que essas funções não são obrigatórias e variam dependendo do contexto no qual ele está inserido e das necessidades da organização ao utilizar um repositório.

- **Identificação e descrição**: para se descrever um componente é possível que a ele seja atribuído um conjunto de características, tais como, nome, domínio, palavras-chave, dentre outras, que os identificam e os diferenciam dos demais ativos que compõem esse mesmo repositório. Para cada componente armazenados deve ser possível identificá-lo homogeneamente dentro de um repositório, ou seja, os componentes de um mesmo tipo devem apresentar o

mesmo conjunto de características. Isso não significa, contudo, que componentes diferentes devam ter os mesmos valores para esse conjunto de características.

- **Inserção**: um Sistema de Gestão de Componente deve permitir que usuários autorizados insiram novos componentes ou ainda, novas versões dos mesmos dentro do repositório. A inserção significa adicionar no repositório o corpo e a descrição do componente.
- **Exploração do catálogo**: aos usuários de um Sistema de Gestão de Componentes, deve ser permitido que explorem o catálogo de componentes para que possam conhecer e analisar as características dos componentes disponíveis.
- **Pesquisa textual**: um Sistema de Gestão de Componentes deve permitir que seus usuários façam pesquisas mais específicas na descrição dos componentes. Como resultados da pesquisa, serão obtidos um ou mais componentes que satisfaçam as condições desejadas. Observados os resultados, pode-se decidir por um maior detalhamento ou generalização dos critérios anteriores.
- **Recuperação**: após a identificação do componente desejado, o Sistema de Gestão de Componentes deve permitir que seus usuários recuperem esse componente do repositório para que possam posteriormente utilizá-lo em um processo de reúso.
- **Organização e pesquisa**: como visto anteriormente, a funcionalidade de exploração de catálogos não é suficiente à medida que a quantidade de componentes disponíveis aumenta. Além disso, a pesquisa textual, muitas vezes, demanda muito tempo. Por causa desses fatores, alguns Sistemas de Gestão de Componentes podem adotar novas formas de organizar as

características dos componentes de modo a permitir que a pesquisa seja baseada em outros critérios.

- **Histórico**: é importante para o gerenciamento de um Sistema de Gestão de Componentes que ele possa armazenar informações de uso, modificações, criação e exclusão de cada um dos componentes disponíveis. Essas informações devem montar uma base histórica que facilitará a análise e a reutilização de componentes.
- **Mensuração**: um Sistema de Gestão pode coletar estatísticas que facilitem o seu gerenciamento. Algumas das principais estatísticas que podem ser adotadas são: frequência de acesso ao Sistema de Gestão de Componente, quantidade de componentes disponíveis, taxa de recuperação, porcentagem de pesquisas bem-sucedidas, frequência com que um determinado componente é analisado ou modificado, dentre outras.
- **Controle de acesso**: um repositório pode adotar uma política de segurança para que determinadas funções só estejam acessíveis a pessoas autorizadas. Por exemplo, pode-se definir para uma empresa específica uma política de segurança onde a pesquisa textual esteja disponível para os funcionários, mas a recuperação de ativos esteja disponível apenas aos funcionários da área de desenvolvimento.
- **Gerenciamento de versões:** um Sistema de Gestão de Componentes pode conter várias versões de um mesmo ativo e, sendo assim, é recomendável que haja algum mecanismo para controlar essas versões e estabelecer o relacionamento entre elas.
- **Controle de modificações**: é recomendável que sejam providas algumas funções para fazer o gerenciamento de modificações dos componentes em um repositório. Essas funções incluem

procedimentos para solicitação de alterações, discussões e permissão para as mesmas. É de extrema importância para a manutenção da consistência de um repositório que quaisquer intenções de modificar os componente sejam comunicadas àquele responsável pela sua administração para que o mesmo, juntamente com outros responsáveis, possa analisar as alterações solicitadas e manter a coerência entre os diversos componentes de software.

- **Notificação de mudanças**: é possível que a qualquer momento um componente seja modificado ou, até mesmo, que o próprio Sistema de Gestão de Componentes sofra algumas alterações em suas principais funções. Sendo assim, um Sistema de Gestão de Componentes pode prover funções que notifiquem seus usuários das modificações recentemente ocorridas, tais como, disponibilização de novas funções, novas políticas de segurança, inserção ou exclusão de componentes, alterações em documentos, entre outras.
- **Acesso pela rede**: como a maioria das empresas estão migrando seus modelos computacionais para modelos centralizados, um requisito típico é que o Sistema de Gestão de Componentes possa acessar o repositório de qualquer ponto em uma rede (Ezran, 1999). Isso é particularmente importante quando seus usuários estão distribuídos geograficamente.
- **Modelo de Negócio**: o repositório pode ser pago ou gratuito.
- **Política de Qualidade**: para qualquer modificação e/ou adição de componentes ocorridas no repositório, deve, juntamente aos componentes alterados/inseridos, ser adicionado documentos que garantem a qualidade do produto.

2.4 Seleção de Componentes

Segundo Resende (2005), a seleção de componentes pode ser considerada como um processo que formaliza a busca por componentes, tanto no mercado quanto dentro da própria organização, bem como a atividade de avaliar se determinado componente, previamente desenvolvido e testado, apresenta-se apropriado para o uso no contexto de um novo sistema.

Kontio (1996), afirma que a falta de um processo para seleção de componentes de software, capaz de reduzir riscos inerentes à aquisição de partes oriundas de diferentes fontes, tem sido uma das principais dificuldades enfrentadas pela comunidade de Engenharia de Software.

Segundo Resende (2006), o fracasso nas seleções de componentes deve-se, dentre outros fatores, a ausência de um processo bem definido para guiar esta ação. Comprar componentes, apesar de tratar-se de uma tarefa até certo ponto rotineira, é uma atividade de alto risco e muitas vezes empírica. Entretanto, apesar do risco de fracasso, muitas organizações não usam procedimentos bem definidos para realizar as suas aquisições.

A seleção de componentes precisa ser realizada, de forma eficiente e com grau de rigor apropriado, para a aplicação em questão. Um processo subjetivo de seleção de componentes fica sujeito a falhas, devido a dificuldade em examinar componentes concorrentes e aparentemente semelhantes. Uma organização imatura em seus processos de aquisição para sistemas de software pode levar um projeto ao fracasso, assim como uma organização com processo de desenvolvimento imaturo (Guerra; Alves, 2004).

2.5 XML

Segundo Pimentel; Teixeira; Santanche (2000), XML (*eXtended Markup Language*) é uma linguagem de marcação apropriada para a representação de dados, documentos e demais entidades cuja essência fundamenta-se na capacidade de agregar informações.

A XML é utilizada para estruturar dados, enquanto o HTML (*Hipertext Markup Language*) é usado basicamente para a apresentação de conteúdo. Neste contexto, é importante ficar explícito que o XML não é um substituto do HTML. O XML é, na realidade, uma maneira de solucionar determinados problemas que são encontrados ao usar-se o HTML.

Segundo Pitts-Moultis; Kirk (2000), com o XML tem-se um melhor controle em relação ao *layout*, um menor esforço no servidor *Web* devido à capacidade de acessar informações do lado do cliente, , a capacidade de publicar qualquer tipo de informação tanto na Internet quanto em intranets e um número menor de problemas para exibir páginas longas.

Pitts-Moultis; Kirk (2000) afirma que utilizando linguagens de marcação estruturadas, como XML, tem-se também maior flexibilidade de pesquisa. É possível, por exemplo, pesquisar apenas informações de cabeçalho ou então pesquisar em cabeçalhos específicos como exemplo, os cabeçalhos de um livro. Não será possível fazer isso se o livro foi criado como um documento HTML. Se o livro inteiro fosse um único documento, seria necessário pesquisar o arquivo inteiro; se fosse um arquivo individual, seria preciso a ajuda de um aplicativo de servidor *Web* separado para pesquisar todos ao mesmo tempo.

Além disso, Pitts-Moults; Kirk (2000) ressalta que o XML fornece uma ampla gama de recursos que não são encontrados no HTML, incluindo uma linguagem extensível que fornece a capacidade de definir suas próprias marcas e atributos. Esses elementos e suas marcas de início e de fim juntamente com seus

atributos o ajudam a definir os elementos estruturais do documento. A capacidade de aninhar estruturas de documentos dentro das estruturas de outros documentos para criar documentos complexos é outro recurso a ser citado.

A W3C (*World Wide Web Consortium*) define os objetivos da XML da seguinte maneira:

- XML deve ser de uso ágil e fácil na internet;
- XML deve suportar uma grande variedade de aplicações;
- XML deve ser compatível com o SGML;
- Os documentos XML devem ser legíveis e de fácil compreensão para os seres humanos;
- A especificação XML deve ser precisa;
- Os documentos XML devem ser fáceis de criar;
- Os documentos XML não precisam ser concisos

2.6 Reusable Asset Specification (RAS)

O RAS (*Reusable Asset Specification*) é um padrão criado pela OMG (*Object Management Group)* usado para descrever bens reutilizáveis (OMG, 2005). O RAS utiliza a tecnologia XML para armazenar seus dados, o que facilita sua extensão uma vez que um arquivo XML é facilmente extensível.

No contexto deste trabalho, é visto frequentemente os termos bens, componentes e artefatos. Artefato, segundo o RAS, é qualquer produto criado no ciclo de desenvolvimento de um software, por exemplo, código-fonte, casos de uso, diagrama de classes entre outros. A Figura 2 mostra a relação entre componentes, bem e artefato.

Um Componente, seja ele abstrato ou concreto, é um Bem. Mas, o contrário nem sempre é verdade. Podem existir bens que são definições de interface ou configurações arquiteturais. Esses bens são materializados em arquivos, que por sua vez representam um ou mais Artefatos. Ou seja, um bem possui um ou mais artefatos. Por exemplo, o código-fonte pode ser um de vários artefatos de um componente concreto (OMG, 2005).

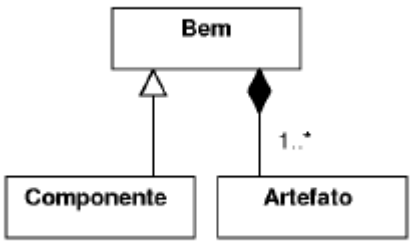

Figura 2 Relação entre Bem, Componente e Artefato.
Fonte (OMG, 2005)

Segundo a OMG (2005), a especificação RAS pode ser dividida em *Core RAS* e *Profiles*. O *Core RAS* descreve os elementos básicos de uma especificação e os *Profiles* descrevem as extensões desses elementos. O *Core RAS* é uma descrição abstrata que é materializada pelo *Asset*. O *Asset* é um elemento que descreve o bem e tem como atributos o nome, id, status, versão entre outros.

A Figura 3 mostra o elemento *Asset*, seus atributos e seus sub-elementos: *Related- Asset*, *Profile*, *Solution*, *Classification* e *Usage*.

- *RelatedAsset* descreve os bens relacionados;
- *Profile* apresenta algumas características do *profile* usado por este bem;

- *Classification* oferece uma descrição mais detalhada sobre o tipo do bem, por exemplo, se é um bem utilizado na área médica;
- *Usage* mostra em qual contexto este bem deve ser usado e;
- *Solution* detalha a solucão.

Apenas os elementos dos dois níveis mais altos estão representados na Figura 3 e os atributos dos sub-elementos de *Asset* também não foram representados para tornar mais fácil o entendimento da figura.

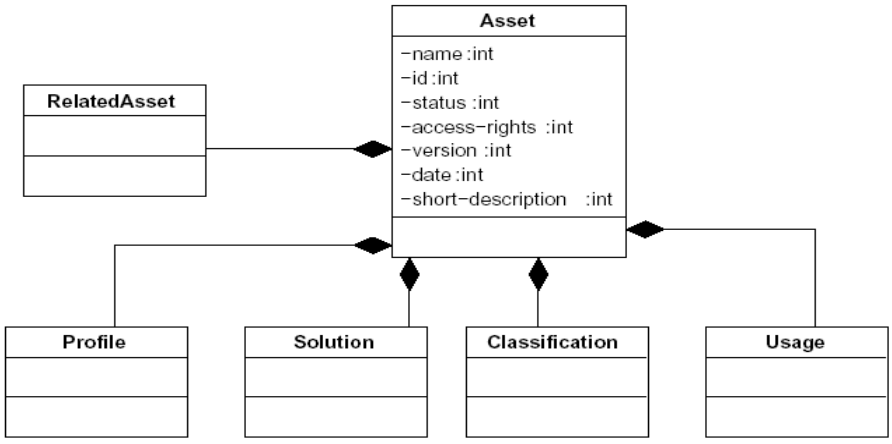

Figura 3 Principais Domínios do *RAS*.
Fonte (OMG, 2005).

A Figura 4 mostra o relacionamento entre os conceitos do *RAS*. O *Core RAS* que é materializado pelo *Default Profile*, que por sua vez é estendido pelos *profiles*. Dois *profiles* aparecem na especificação do RAS, o *Default Component Profile* e o *Default WebServices Profile*. Entretanto, outros *profiles* podem ser criados se algum conceito adicional sobre um bem deve ser especificado.

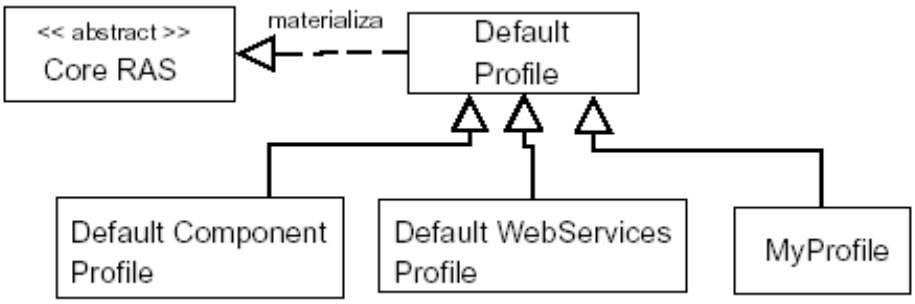

Figura 4 Core RAS e os *Profiles*.
Fonte (OMG, 2005)

3 PRINCIPAIS SISTEMAS DE GESTÃO DE COMPONENTES

Nesta seção, apresentam-se os principais Sistemas de Gestão de Componentes (SGC) selecionados para análise comparativa. A seguir faremos uma breve descrição de cada SGCs.

3.1 *Agora*

Segundo Seacord (1999), ***Agora*** é um Sistema de Gestão de Componentes desenvolvido pelo *Software Engineering Institute* (SEI), cujo objetivo é facilitar a busca por componentes usando uma base de dados distribuída, gerada e indexada automaticamente e de amplitude mundial.

O ***Agora*** utiliza introspecção combinada com o mecanismo *Web AltaVista Internet Service* para procurar, analisar e indexar componentes disponíveis na internet. Os produtos de software encontrados são classificados por modelos de componentes.

A introspecção é um termo associado primeiramente com *JavaBeans* (componentes de software escritos na linguagem de programação Java) que descreve a capacidade dele para fornecer informações sobre suas próprias interfaces.

Agora suporta dois processos básicos: a localização e a indexação e a busca e a restauração. A localização e a indexação de componentes no ***Agora*** são feitas usando agentes de software, que vasculham a Internet para descobrir e coletar recursos, enquanto o usuário busca e restaura os componentes.

A base de componentes do ***Agora*** é distribuída, ou seja, os componentes são armazenados na *Web*, e não dentro do repositório. Ao realizar pesquisas, referências URL (*Uniform Resource Locator*) das localizações dos componentes na Internet são retornadas. O armazenamento de componentes é

responsabilidade do produtor do componente, o ***Agora*** apenas mantém referências para os componentes.

O ***Agora*** usa uma variedade de agentes para localização e indexação das informações dos componentes. Para isto foi desenvolvido um agente *JavaBeans* e o *CORBA* cada um com a capacidade de localizar e indexar componentes do seus respectivos tipos.

Componentes são introspectados durante a fase de indexação para descobrir suas interfaces. A introspecção do JavaBeans é realizada usando o mecanismo promovido pela classe introspectiva do JavaBeans.

Uma vez que um componente foi identificado, a informação da interface é decomposta em um conjunto de *tokens*. Para incluir os *tokens* ao índice criado pelo agente responsável por localizar e indexar é criado um documento. Diferentemente de um documento de texto, a informação da interface do componente pode ser diferenciada em campos distintos, exemplos de campos podem ser métodos, atributos, ou eventos. Esta informação é mantida para cada componente com o intuito de permitir a realização de buscas especificas a serem realizadas. O nome e o tipo do componente são sempre preservados como campos para permitir buscas pelo nome e pelo tipo de componente. Meta-informação sobre cada componente é sempre mantida com o documento, incluindo a URL para cada componente. Mantendo o URL do componente, permite que a informação detalhada da interface volte a ser coletada durante o processo de busca e recuperação e permite ao usuário examinar neste caso, o local onde o componente realmente se encontra.

A busca e a restauração no ***Agora*** é um processo de duas etapas. Inicialmente, a busca é realizada por palavra-chave e, opcionalmente, especifica o tipo de componentes. Estes termos e outros critérios são pesquisados nos índices coletados pelos agentes de pesquisa. É possível também realizar pesquisas em um contexto particular (por exemplo, encontrar componentes com

um determinado nome ou componentes que implementam um dado modelo de componente). O resultado da pesquisa é enviado de volta para o usuário por inspeção. Cada resultado inclui meta-informação incluindo o URL do componente. O pesquisador pode refinar ou ampliar o critério de pesquisa baseado no número e na qualidade de correspondências.

Após o pesquisador ter completado sua busca, os componentes podem ser examinados individualmente com detalhes. A URL para o componente é retornada para outra instrospecção. Este processo reduz o numero de informações que são mantidas no índice para cada componente indexado, e garante que o componente ainda esteja disponível no local especificado, ou seja, na URL retornada.

A Figura 5 mostra o resultado de uma pesquisa utilizando a busca por *JavaBeans*. Neste caso, os critérios de pesquisa especificam que o *JavaBeans* deverá conter os métodos *color* e *draw* e a propriedade *color*, mas não deverá conter o termo funscroll. Essa pesquisa resultou dois documentos, cada um com ranking de relevância relativamente baixo. A contagem da palavra indica quantas ocorrências de cada termo foram encontradas no banco de dados.

Figura 5 Interface de Consulta do ***Agora***
Fonte (Seacord, 1999)

Depois de uma lista de resultados encontrados, a pesquisador poderá selecionar o *link* para a URL do componente. Isto normalmente permite para o pesquisador ver como o componente é, quando ele é operável e possivelmente coletar informações adicionais sobre o componente. Selecionando o nome do componente, abrirá uma interface de descrição do componente.

Agora permite o uso dos operadores '+' e '-': estes operadores são responsáveis por filtrar o resultado da pesquisa. ***Agora*** também permite a capacidade de realizar buscas avançadas que suporta operadores lógicos como AND, OR, NOT e NEAR.

Agora suporta um número de funções especiais que permite ao usuário estreitar o critério de pesquisa, ou seja, focalizar cada vez mais o rumo da pesquisa, usando características específicas de cada componente. O Quadro 1 seguinte mostra funções especiais que operam através de todos os tipos de componentes.

Quadro 1 Funções Especiais *JavaBeans* para Todos Tipos de Componentes.

Palavra-chave	Descrição
Componente:tipo	Encontra componente deste tipo, cada tipo possui alguns agentes
Nome:nome	Encontra componentes o nome passado

3.2 *Spars-J*

Segundo Yokomori; Ishio; Yamamoto; Matsushita; Kusumoto; Inoue (s.d), ***Spars-J*** é um Sistema de Gestão de Componentes responsável por armazenar, analisar e recuperar componente Java. É fácil imaginar que programas similares têm sido desenvolvidos em diferentes lugares do mundo e em diferentes tempos, sem nenhum compartilhamento de conhecimento sobre seus programas. Considera-se que uma boa coleção de programas ou componentes pode aumentar a produtividade do desenvolvimento e da qualidade do software que está sendo desenvolvido.

Segundo Yokomori; Ishio; Yamamoto; Matsushita; Kusumoto (s.d), ***Spars-J*** pode ser considerado como o sistema Google para os engenheiros de software. Neste sistema, vários códigos fonte Java são coletados e armazenados em um arquivo. Estes componentes são classificados por sua avaliação (chamada de *ComponentRanks*), na qual é determinada por suas relações de uso. Um pesquisador de componentes que queira saber sobre a definição ou o uso de algum componente deve realizar uma consulta por palavras-chave com o nome do componente desejado. Esta consulta é analisada e os resultados são listados pela classificação do componente. Pelo *Component Ranks* , é considerado que componentes com alta reusabilidade podem ser encontrados facilmente.

Spars-J, consiste em dois subsistemas, o primeiro é responsável por construir a base de dados e o segundo é responsável por pesquisar componentes.

No primeiro subsistema, uma base de dados para busca de componente é construída a partir de arquivos de código fonte Java. Após coletados, diversos programas fonte Java são analisados sistematicamente e armazenados em um

arquivo com informações, como nomes das classes, métricas de cada componente e palavras-chaves. As palavras-chaves são indexadas para fazer uma espécie de dicionário para quando o usuário inserir a palavras chave no campo de pesquisa, o sistema vai até o dicionário e pesquisa pela palavra em questão e, logo em seguida, retorna valores, caso exista. O uso dos nomes da classe é analisado para determinar as relações entre os componentes. As métricas são usadas para medir a similaridade entre os componentes. No processo de desenvolvimento de software, nós pode-se imaginar que um componente pode ser reusado com mudanças pequenas, e não simplesmente copiando. Para calcular os componentes como um grupo, este Subsistema calcula similaridades entre os componentes. Como um exemplo de valor métrico, é usado LOC (*Line-of-Code*), complexidade ciclomática e assim por diante. Componentes similares são empacotados em um mesmo grupo e as relações de uso associadas são fundidas. Todos os componentes (Grupos) são classificados pelo *Componente Ranks* no qual são determinados pela sua relação de uso, e a classificação de cada componente é então armazenada em um arquivo.

Por outro lado, o subsistema Pesquisar Componentes, oferece uma função de recuperação para o usuário. Basicamente, este subsistema executa a recuperação dos componentes que foram retornados a partir de uma determinada palavra-chave, inclusive os comentários em arquivos de código-fonte. Entretanto, o usuário pode optar por realizar uma busca avançada de acordo com sua necessidade. Por exemplo, não incluir no resultado da pesquisa com o qual a palavra-chave aparece somente em seu comentário, ou somente listar os componentes no qual a palavra chave aparece na definição da classe ou método.

A consulta realizada pelo usuário é analisada e decomposta em uma série de palavras-chave. Para cada palavra-chave, o subsistema verifica o componente com o qual a palavra-chave aparece no 'dicionário'. O resultado é ordenado pelo *ComponenteRank* e o usuário recebe o resultado pelo navegador

com a informação de cada componente (Figura 6). Além disso, conforme ilustrado na Figura 7, o usuário pode clicar nos resultados para obter mais detalhes sobre o componente, facilitando o processo de recuperação.

Figura 6 Retorno de uma Pesquisa Realizada pelo Usuário

eclipse.swt.layout.GridData

Source Code | Group | Using GridData | Used by GridData | Metrics | File Information | Download

Go to the class start line
Go to the first highlight of **grid**

```
/*******************************************************************************
 * Copyright (c) 2000, 2005 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package org.eclipse.swt.layout;

import org.eclipse.swt.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.*;

/**
 * <code>GridData</code> is the layout data object associated with
 * <code>GridLayout</code>. To set a <code>GridData</code> object into a
 * control, you use the <code>Control.setLayoutData(Object)</code> method.
 * <p>
 * There are two ways to create a <code>GridData</code> object with certain
 * fields set. The first is to set the fields directly, like this:
 * <pre>
```

Figura 7 Informações sobre o Componente Selecionado

3.3 *Sensedia*

Sensedia é uma empresa que oferece soluções para reutilização de software e governança SOA (*Service-Oriented Architecture*). Com uma oferta liderada pelo SGCs ***Sensedia***, a empresa ajuda seus clientes a alcançarem benefícios como redução de custos de desenvolvimento e aumento da agilidade na construção de softwares, além de reduzir o desenvolvimento duplicados, no qual possibilita produtos mais maduros e adequados para a necessidade do mercado. Os serviços também incluem programas de reúso de software, identificação de componentes e serviços, repositórios de metadados e ferramentas de apoio aos processos de criação, reutilização, governança e análise de qualidade de componentes de software.

O ***Sensedia*** é um SGCs responsável por centralizar informações sobre serviços SOA, componentes e outros ativos criados ao longo do ciclo de desenvolvimento. O SGCs é uma peça chave dentro da iniciativa SOA, pois provê um ambiente integrado para a governança desses serviços.

Alguns dos maiores desafios na adoção de uma estratégia SOA e reúso são:

- Baixa visibilidade do acervo digital e de serviços, tanto dos serviços planejados quando dos serviços e ativos já existentes;
- Crescente complexidade nas áreas de negócio e, principalmente, nas áreas de TI (Tecnologia da Informação);
- Dificuldade para gerenciar os impactos de mudanças.

O ***Sensedia*** foi concebido levando em consideração os desafios listados acima.

Como elemento central na estratégia de mudanças fundamentais em TI e negócios, o ***Sensedia*** possibilita a mensuração de diversos resultados:

- Governança dos ativos, aumentando a visibilidade e o mapeamento entre os diversos elementos do repositório – componentes, processos de negócio, *frameworks*, serviços etc.;
- Controle e organização mesmo em ambientes heterogêneos de TI;
- Esforços guiados em direção as melhores práticas de programas SOA;
- Abordagem prescritiva para otimização sistemática do processo de reúso, padronização e consolidação de ativos.

O SGCs ***Sensedia*** é baseado em padrões abertos e bastante flexíveis quanto aos requisitos necessários para sua implantação, podendo ser rodado nos

sistemas operacionais Windows, Linux e Unix. Na parte do usuário, pode ser usado tanto no Internet Explorer (Microsoft) quanto no Firefox (Mozilla).

Com uma interface bem amigável, o ***Sensedia*** permite ao usuário encontrar e resgatar um determinado componente. Cada usuário possui um *login* e senha necessários para entrar no sistema e, logo após a entrada, a interface principal que também é chamada de *Home* (Figura 8) é apresentada para o usuário. Na página *Home*, podem ser encontradas as ferramentas que o sistema possui como buscador, cadastro de componentes (visto somente na visão do administrador) e informações úteis como novidades, destaque, notícias.

Em seu sistema de busca, o ***Sensedia*** possui o mecanismo de *auto-suggest*, cálculo de relevância associado aos resultados e busca por nuvens de tags, indexando automaticamente ativos e artefatos relacionados (incluindo o conteúdo de PPTs. PDF's, DOC's, etc), podendo ser ordenados de diferentes formas e filtrados dinamicamente. Depois de selecionado um componente, o ***Sensedia*** mostra detalhes como: atributos associados ao tipo, histórico, estatística, meta-informação customizáveis, relacionamentos, artefatos associados e *tags* relacionadas. O sistema permite a visualização dos relacionamentos de forma gráfica facilitando a analise de impacto, a navegação entre ativos e o cálculo de ROI (R*eturn on Investment*).

O ***Sensedia*** possui integração com ambiente de desenvolvimento. *Plug-ins* para IDEs (*Integrated Development Environment*) como Eclipse, JDeveloper e VisualStudio no qual permite ao usuário desenvolver e, ao mesmo tempo, buscar por componentes usando palavras-chave, analisar as características dos componentes listados, versão em que se encontra, relacionamento com outros componentes, entre outras.

Sensedia é um Sistema de Gestão de Componentes que incentiva e ajuda a prática do reúso, facilitando na construção de softwares e, ao mesmo tempo, garantindo um software mais maduro.

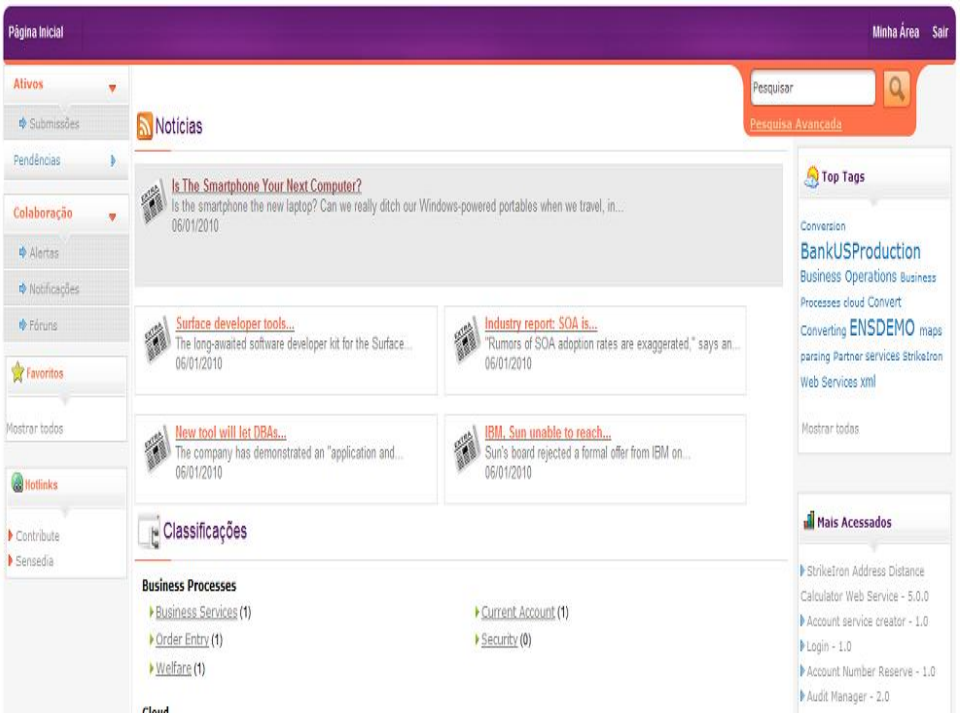

Figura 8 Página Principal (*Home*) do ***Sensedia***

3.4 *ComponentSource*

ComponentSource (2010) é um mercado de opções destinado àqueles responsáveis por especificar, localizar e obter componentes de software. Esse mercado está aberto por meio de uma política própria, à inserção dos mais diversos componentes independentemente de autores. O ComponenteSource disponibiliza aos usuários algumas vantagens, tais como, avaliação gratuita de versões, suporte técnico, atendimento em vários idiomas (inclusive português), centro de soluções para construção de componentes e para comércio eletrônico.

Totalmente voltado para web, cada usuário do sistema possui *login* e senha necessários para ter acessos aos componentes desejados. Cada

componente possui um tipo de licença e cada licença possui um valor agregado, escolhida de acordo com a necessidade da empresa/usuário. Após a entrada no sistema, na interface principal (Figura 9), são listados os produtos em destaque, assim como produtos recém chegados ao repositório. O catálogo de componentes pode ser explorado usando categorias, autores ou em ordem alfabética.

A pesquisa, realizada usando palavras-chave, pode ser restringida a um determinado tipo de componente (Java, COM, Visual Basic, Delphi ou Business). Depois de realizada a pesquisa, o sistema por padrão lista os componentes ordenados por sua relevância, porém o usuário pode escolher outras formas de ordenação: mais vendidos, mais comentados e em ordem alfabética. Para cada componente requisitado, existem informações técnicas de utilização, preço, compatibilidade, pré-requisitos, licença de uso, descrição, dentre outras, facilitando o resgate do componente que melhor atende as necessidades do pesquisador dentro do repositório.

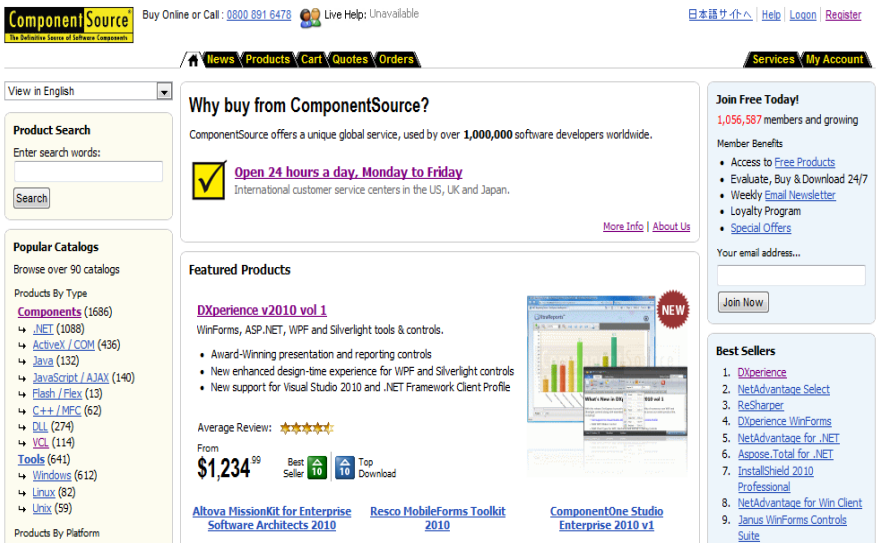

Figura 9 Página comercial do ***ComponentSource***

3.5 *CodeBroker*

CodeBroker é um Sistema de Gestão de Componentes, que roda por trás do ambiente de desenvolvimento (Emacs) e utiliza a entrega de informação ativa para:

- Mostrar ao desenvolvedor, componentes que tem maior probabilidade de serem usados em suas tarefas de desenvolvimento atual (Yunwen; Fischer; Reeves, 2000);
- Redução do custo global do processo de localização do componente eliminando o passo de criação de consultas (Yunwen; Fischer; Reeves, 2000);
- Eliminação da necessidade do desenvolvedor de trocar contextos entre o ambiente de desenvolvimento e o sistema de repositório de componentes (Yunwen; Fischer; Reeves, 2000).

A arquitetura do ***CodeBroker*** é mostrada na Figura 10. Ela consiste em três agentes: *Listener*, *Fetcher* e *Presenter*. Segundo Bradshaw; Dumais (1997) um agente de software é uma entidade de software que funciona de forma autônoma em resposta às mudanças no seu ambiente de execução sem exigir orientação ou intervenção humana. No ***CodeBroker***, o *Listener* extrai e formula consultas de reúso monitorando a interação do desenvolvedor com o editor. Estas consultas são passadas para *Fetcher* que recupera os componentes correspondentes do repositório de reúso. Os componentes recuperados pelo *Fetcher* são passados para o *Presenter*, que usa o perfil do usuário para filtrar e retirar os componentes indesejados, e os demais são listados no *Reusable Components Info-display* (*RCI-display*), colocado sobre a janela do editor.

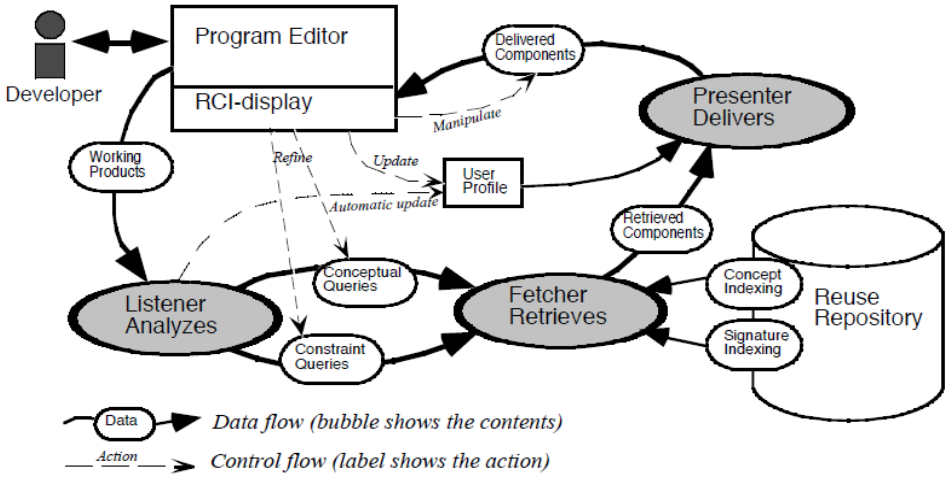

Figura 10 Arquitetura do ***CodeBroker***.
Fonte Yunwen; Fischer; Reeves (2000)

O repositório de reúso no ***CodeBroker*** é criado pelo programa indexador, *CodeIndexer*, que extrai e indexa descrições e assinaturas funcionais das documentações *online* geradas pelo *running javadoc* em códigos fonte Java. Cada componente e indexado em duas formas: indexação de conceito e indexação de assinatura. Indexação de conceito associa componentes baseados em sua descrição funcional em textos e a indexação de assinatura cria uma assinatura de representação para cada componente.

3.5.1 Listener

O agente *Listener*, roda continuamente por trás do Emacs para monitorar as entradas do desenvolvedor. Seu objetivo é extrair e formular consultas de reúso baseado nas "pistas" presente no ambiente de desenvolvimento. Consultas são automaticamente extraídas dos comentários e assinaturas. A Figura 11 mostra um exemplo no qual o desenvolvedor quer gerar um número aleatório entre dois

inteiros, logo após a inserção do comentário, o agente Listener captura o conteúdo do comentário e retorna os componentes associados a ele.

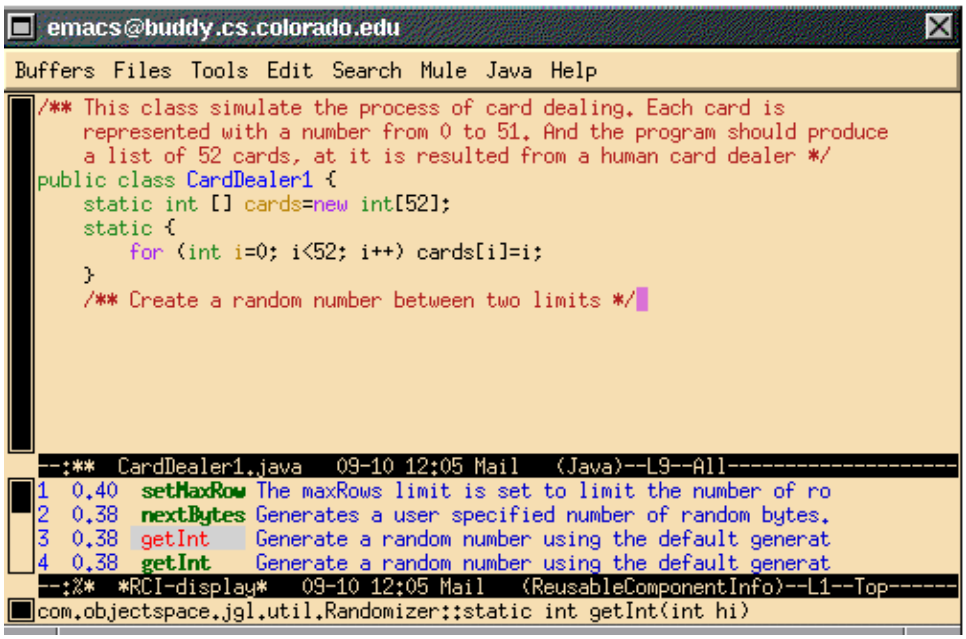

Figura 11 Retorno de Componentes Usando Comentário.
Fonte Yunwen; Fischer; Reeves (2000)

3.5.2 Fetcher

O mecanismo de reposição usado pelo *Fetcher* é a combinação da Análise Semântica Latente (ASL) proposto por Landauer; Dumais (1997) e Assinatura de Correspondência (AC) proposta por Zaremski (1995). ASL é usada para calcular a similaridade entre consultas e descrições funcionais de componentes reutilizáveis. AC é usada para determinar a restrição de compatibilidade entre consultas e as assinaturas dos componentes reutilizáveis.

Segundo Yunwen; Fischer; Reeves (2000), ASL é uma tecnologia baseada em indexação de textos-livres. Texto-livre sofre do problema de

recuperação baseado em conceito. Se desenvolvedores usam termos diferentes dos usados nas descrições dos componentes, eles não conseguem encontrar o que querem, porque a indexação de texto-livre não leva em consideração a semântica. Ao construir um grande espaço semântico dos termos para captar o padrão geral das suas relações associativas, o conceito básico ASL facilita a recuperação e preenche a lacuna conceitual na formulação de consultas para a reutilização.

Segundo Landauer; Dumais (1997), AC é o processo de determinar a compatibilidade de dois componentes em relação a suas assinaturas.

3.5.3 Presenter

Os componentes retornados são então mostrados para o desenvolvedor pelo agente *Presenter* no *RCI-display* em ordem decrescente de similaridade. Cada componente é acompanhado com seu valor de similaridade, seu nome e uma curta descrição (Figura 12). Os desenvolvedores interessados em um determinado componente podem executar, por um clique do mouse, uma interface de navegação externa, o que poderia ser qualquer *browser* HTML com texto, para ir ao local correspondente da documentação completa Java.

O agente *Presenter* usa perfis de usuário para adaptar os componentes obtidos para o nível de conhecimento de cada desenvolvedor. O perfil de usuário é um arquivo listando todos componentes conhecidos pelo desenvolvedor. Um componente é dito conhecido quando já foi reutilizado anteriormente. Cada item da lista pode ser um pacote, uma classe ou um método.

- Um pacote indica que nenhuma classe ou método serão listado no *RCI-Display;*

- Uma classe indica que nenhum método deve ser listado *RCI-Display*;
- Um método indica que apenas métodos devem ser listados *RCI-Display*.

Os perfis de usuário são automaticamente atualizados pelo *Listener*. Se o *Listener* observar que o usuário reutilizou um componente diversas vezes, ele assume que não há mais necessidade de listar este componente, porque se ele é reusado, o *Listener* considera que o desenvolvedor recorda deste componente e adiciona em seu perfil.

O Perfil do usuário pode ser editado manualmente ou utilizando a interação com o ***CodeBroker***. Um clique com o botão direito do *mouse* em cima do componente retornado pelo *Presenter* abre o menu do componente, como mostrado na Figura 12. O desenvolvedor pode selecionar o comando *All Sessions*, o qual irá atualizar seu perfil, e o componente ou os componentes desta classe ou deste pacote não serão mais listados como resultado na próxima sessão. Caso o *Presenter* liste um componente que não seja interessante para o usuário, ele poderá removê-lo desta sessão usando o comando *This Session Only*. Isto ocorre para remover componentes não interessantes para a tarefa em questão e facilitar a busca pelo componentes requeridos.

O comando *Query Refinement* traz ao desenvolvedor a possibilidade de refinar a consulta, onde ele pode modificar a consulta que foi feita automaticamente e executar uma nova localização.

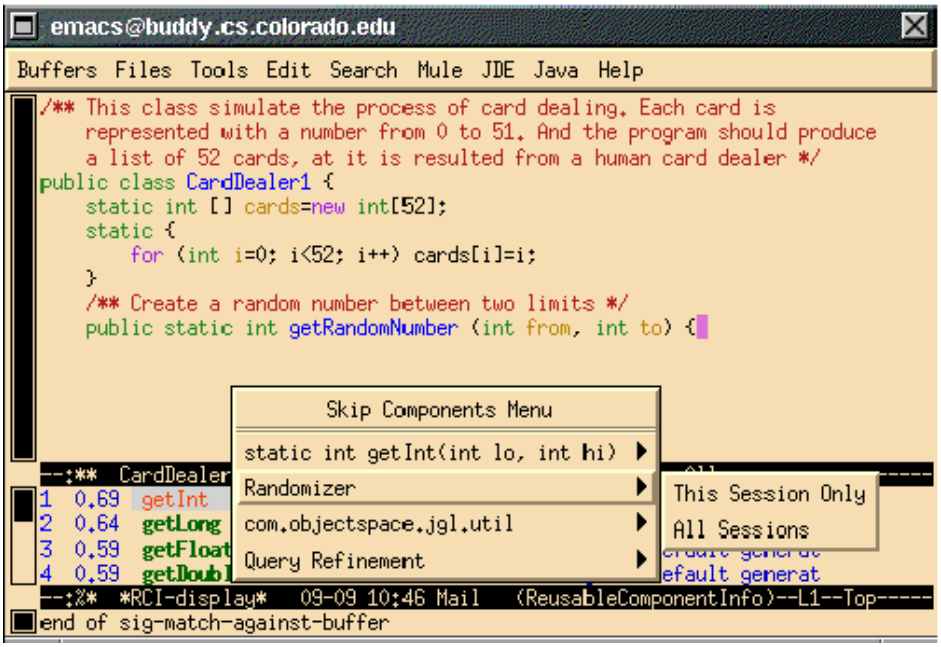

Figura 12 *Skip Components Menu*, Menu de Opções para cada Componente Listado

Fonte Yunwen; Fischer; Reeves (2000)

4 METODOLOGIA

Segundo Zambalde A. L., Pádua S. P. I. C., Alves R. M. (2008), metodologia (ou material e métodos) deve trazer a descrição do tipo de pesquisa e dos procedimentos metodológicos adotados para o desenvolvimento da mesma, ou seja, uma descrição breve, porém completa e clara das técnicas e processos empregados, bem como do delineamento experimental.

Com relação ao "Tipo de pesquisa", Jung (2004) e Marconi; Lakatos (2003) apud Zambalde A. L., Pádua S. P. I. C., Alves R. M. (2008) afirmam que ela pode ser classificada quanto a sua natureza (básica ou fundamental / aplicada ou tecnológica); quanto aos seus objetivos (exploratória / descritiva ou / explicativa); quanto aos procedimentos (experimental / operacional / estudo de caso) e quanto ao local de realização da mesma (laboratório ou campo).

Este trabalho, seguindo a classificação acima, seria de natureza aplicada. Silva e Menezes (2000) afirmam que, o objetivo da pesquisa aplicada é gerar conhecimentos para que seja possível a aplicação prática dirigidos à solução de problemas específicos. Entende-se que a pesquisa aplicada é adequada para a comparação entre Sistemas de Gestão de Componentes de Software, a fim de gerar uma tabela comparativa.

Em relação aos objetivos, a presente pesquisa pode ser classificada como do tipo descritiva. Uma pesquisa descritiva, segundo Gil (1991), tem o objetivo de descrever as características de certa população, fenômeno, ou o estabelecimento de relações entre variáveis. Além disto, envolve o uso de técnicas padronizadas de coleta de dados como questionário e observação sistemática. Em geral, assume a forma de Levantamento.

Em relação aos procedimentos técnicos, a pesquisa pode ser caracterizada como estudo de caso. Segundo Gil (1991), estudo de caso envolve

o estudo profundo e exaustivo de um ou poucos objetos de maneira que se permita o seu amplo e detalhado conhecimento.

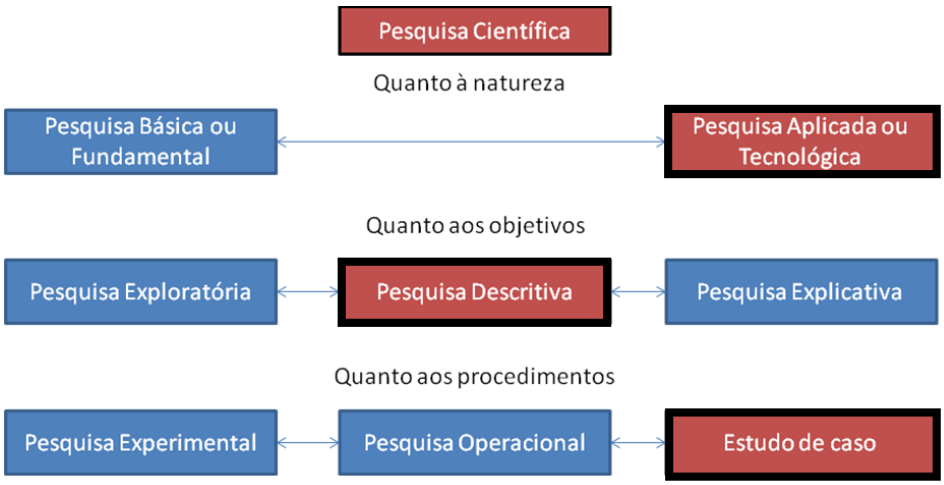

Figura 13 Classificação da Pesquisa Científica

5 ANÁLISE COMPARATIVA

Apresentam-se neste capítulo, os critérios de comparação, as comparações realizadas e os resultados obtidos.

5.1 Critérios de Comparação

Para a execução do presente trabalho foram selecionados alguns critérios que são considerados importantes. Dentre os critérios selecionados encontram-se: Identificação e Descrição, Inserção, Exploração do Catálogo, Recuperação, Histórico, Controle de Acesso, Controle de Versões, Controle de Modificações, Notificação de Mudanças, Modelo de Negócio, Tipo de Componente, Tipo de Suporte.

A seguir são explanados os critérios selecionados para a comparação e sua importância para a pesquisa.

- **Identificação e Descrição**: Define-se Identificação e Descrição como um conjunto de características tais como nome, domínio, palavra-chave responsável por descrever e identificar um componente dentro do Repositório. Ele é importante para que cada componentes contido no Repositório possa ser identificado homogeneamente,ou seja, os componentes de um mesmo tipo devem apresentar o mesmo conjunto de características. O Sim indica que o SGC possui Identificação e Descrição enquanto, o Não indica que o SGC não possui Identificação e Descrição;
- **Inserção**: Define-se Inserção a capacidade de adicionar novos componentes ao repositório. A Inserção é importante para o crescimento do numero componentes dentro do Repositório. O Sim

indica que o SGC possui Inserção, enquanto o Não indica que o SGC não possui Inserção;

- **Exploração do Catálogo**: Define-se Exploração do Catálogo a capacidade do usuário poder conhecer e analisar as características dos componentes disponíveis. Ela é importante por facilitar o usuário na escolha de componente. O Sim indica que o SGC permite Exploração do Catálogo, enquanto o Não indica que o SGC não permite a Exploração do Catalogo;
- **Recuperação**: Define-se Recuperação a capacidade de recuperar componentes dentro de um Repositório para que possam ser utilizado posteriormente em um processo de reúso. Ela é importante para o usuário poder recuperar componentes do Repositório. O Sim indica que o SGC permite a Recuperação, enquanto o Não indica que o SGC não permite a Recuperação;
- **Histórico**: Define-se Histórico como a capacidade de armazenar informações de uso, modificações, criação e exclusão de cada componente contido no repositório. Ele é importante por facilitar a análise e a reutilização de componentes. O Sim indica que o SGC possui Identificação e Descrição, enquanto o Não indica que o SGC não possui Identificação e Descrição;
- **Controle de Acesso**: Defini-se Controle de Acesso a política de segurança adotada pelo SGC. Ele é importante para garantir que determinadas funções só sejam realizadas por usuários que possuem a permissão. Por exemplo, pode-se definir a um usuário a política de segurança onde a adição, modificação e exclusão de componentes estejam disponíveis apenas a ele. O Sim indica que o SGC possui Controle de Acesso, enquanto o Não indica que o SGC não possui Controle de Acesso;

- **Controle de Versões**: Defini-se Controle de Versões a capacidade do SGCs controlar cada versão de componente contido em seu repositório. Ele é importante para garantir que a ultima versão do componente é a que esta no repositório. O Sim indica que o SGC possui Controle de Versões, enquanto o Não indica que o SGC não possui Controle de Versões;
- **Controle de Modificações**: Defini-se Controle de Modificações a capacidade de controlar as requisição de modificação, adição e remoção de componentes dentro do repositório. É de extrema importância para a manutenção da consistência de SGCs que quaisquer intenções de modificar os componentes sejam comunicadas ao responsável pela sua administração para que o mesmo possa analisar as alterações solicitadas e manter a coerência entre os diversos componentes de software contidos no repositório. O Sim indica que o SGC possui Controle de Modificações, enquanto o Não indica que o SGC não possui Controle de Modificações;
- **Notificação de Mudanças**: Define-se Notificação de Mudanças a capacidade de informar qualquer mudança ocorrida no SGC. Ela é importante para manter o usuário atualizado nas modificações ocorridas no Sistema de Gestão de Componentes, informando ao usuário mudanças como Inserção, Modificação e Exclusão de componentes de software. O Sim indica que o SGC possui Notificação de Mudanças, enquanto o Não indica que o SGC não possui Notificação de Mudanças;
- **Modelo de Negócio**: Define-se Modelo de Negócio, a política utilizada pelo Sistema de Gestão de Componentes, podendo ser pago ou gratuito.

- **Tipo de Componentes**: Define-se Tipo de Componente, como a tecnologia na qual foi desenvolvido. Por exemplo, um componente pode ser feito em Java, C++, .NET , etc. Sua importância é informar ao usuário qual tipo de componentes é suportado pelo Sistema de Gestão de Componentes.
- **Tipo do Suporte**: Define-se Tipo do Suporte, o tipo de acesso ao Sistema de Gestão de Componentes, podendo ser *web* ou local. É importante para o usuário do Sistema de Gestão de Componentes saber qual é a forma de acesso ao Sistema, para que o mesmo possa dispor dos recursos necessários para a utilização.

5.2 Aplicação dos Critérios

A seguir mostraremos quadros comparativos entre os Sistemas de Gestão de Componentes de Softwares mencionados no Capítulo 4. Para que a comparação pudesse ser realizada, foram levadas em consideração os critérios listados na sessão 5.1

5.2.1 Identificação e Descrição

Todos os SGCs analisados possuem identificação e descrição dos componentes presentes em seu Repositório. A identificação é feita por palavras-chave e a descrição é atribuída ao componente no momento de sua inserção no Repositório. Os SGCs ***Spars-j***, ***Codebroker*** e ***Agora*** possuem uma característica diferente de palavras-chave dos demais. Por se tratar de SGCs exclusivo para a tecnologia Java, as palavras-chave são os nomes das classes, métodos e pacotes, dependendo do tipo de componente que esteja procurando. Nos SGCs ***Sensedia*** e ***ComponentSource***, a identificação é atribuída ao componente no momento da

inserção, pelo responsável por inserir componente no Repositório. Em todos os SGCs, a descrição também é uma forma de identificação, porém não exclusiva.

Segue o Quadro 2, mostrando a comparação dos SGCs em relação a Identificação e Descrição.

Quadro 2 Análise Comparativa em Relação à Identificação e Descrição.

	SPARS-J	SENSEDIA	COMPONENT E SOURCE	AGORA	CODEBROKER
Identificação	Palavra-chave	Palavra-chave	Palavra-chave	Palavra-chave	Palavra-chave

5.2.2 Inserção

Todos os Sistemas de Gestão de Componentes (SGCs) analisados permitem a inserção de novos componentes juntamente com sua descrição conforme descrito acima.

Segue o Quadro 3, mostrando a comparação dos SGCs em relação a Inserção.

Quadro 3 Análise Comparativa em Relação à Inserção de Componentes

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Permite inserção	Sim	Sim	Sim	Sim	Sim

5.2.3 Exploração do Catálogo

Todos os Sistemas de Gestão de Componentes (SGCs) analisados permitem a exploração do catálogo de componentes.

SGCs como ***ComponentSource*** e ***Sensedia*** permitem a exploração do catálogo de diversas formas, entre elas: por plataforma, por categoria, pelos mais

recuperados, após uma consulta por palavras-chaves, etc. Os demais só permitem a exploração do catálogo após a consulta por palavra-chave.

Segue abaixo o Quadro 4, mostrando a comparação dos SGCs em relação a Exploração do Catálogo.

Quadro 4 Análise Comparativa em Relação à Exploração do Catálogo

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Permite explorar o catálogo de componentes	Sim	Sim	Sim	Sim	Sim

5.2.4 Recuperação

Todos os SGCs analisados permitem a recuperação de componentes. Porém, o ***ComponentSource*** difere dos demais por adotar uma política de restauração paga, ou seja, para que o usuário recupere um determinado componente ele deve pagar por isto e o preço esta vinculado na licença adquirida pelo usuário. O processo de recuperação nos demais repositórios é gratuito.

Segue abaixo o Quadro 5, mostrando a comparação dos SGCs em relação a Restauração.

Quadro 5 Análise Comparativa em Relação ao Processo de Restauração de Componentes

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Recuperação	Sim	Sim	Sim	Sim	Sim
Política de Restauração Adotada	Gratuita	Gratuita	Paga	Gratuita	Gratuita

5.2.5 Histórico

No Sistema de Gestão de Componente (SGC) ***Sensedia***, cada componente possui informações de criação, modificação e remoção. Para componentes recém inseridos no repositório, é lançado, na sessão novidades da página principal, um informativo do novo componente inserido no repositório. Após qualquer modificação de um componente, é inserido junto a ele informações sobre suas modificações que podem ser visualizadas usando o histórico de modificações, contido nas propriedades do componente. O usuário pode optar por receber informações via *e-mail* para qualquer modificação e/ou exclusão de um determinado componente específico.

No Sistema de Gestão de Componente ComponenteSource, somente é informado a inserção de novos componentes, que pode ser visualizada no *site* do repositório ou por notificação via *e-mail* (opção selecionada pelo cliente no momento do cadastro no *site*). Exclusão e modificações não são informadas aos usuários, ou seja, não há um histórico de modificações.

Nos demais repositórios (***Spars-J***, Agora e ***CodeBroker***) não há histórico de modificações.

Segue abaixo o Quadro 6, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Histórico.

Tabela 6 Análise Comparativa em Relação ao Histórico

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Histórico de Inserção	Não	Sim	Sim	Não	Não
Histórico de Modificação	Não	Sim	Não	Não	Não
Histórico de Remoção	Não	Sim	Não	Não	Não

5.2.6 Controle de Acesso

Dos Sistemas de Gestão de Componentes analisados, apenas o ***Agora*** não possui controle de acesso. Por ser um Sistema de Gestão de Componentes no qual seus componentes ficam totalmente distribuídos, ou seja, seus componentes não ficam diretamente em um repositório, mas em vários locais da Web. O Sistema de Gestão de Componentes ***Agora*** não possui controle de inserção, remoção e modificação de seus componentes.

Os Sistemas de Gestão de Componentes ***Spars-J***, ***Sensedia***, ***ComponentSource*** e ***CodeBroker***, possuem sistema de controle de acesso. Para ter acesso a estes repositórios, com exceção do ***ComponentSource***, o usuário é obrigado a fornecer um *login* e senha no momento de utilização do Sistema de Gestão de Componentes.

O Sistema de Gestão de Componentes ***ComponentSource*** permite ao usuário pesquisar seu repositório sem estar logado no sistema, porém para resgatar um componente, o usuário é obrigado a logar no sistema.

As manutenções no repositório como adição de novos componentes, remoção de componentes, modificação de componentes, podem ser realizadas apenas pelo administrador do Sistema de Gestão de Componentes.

Segue abaixo o Quadro 7, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Controle de Acesso.

Quadro 7 Análise Comparativa em Relação ao Controle de Acesso

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Controle de acesso	Sim	Sim	Sim	Não	Sim

5.2.7 Controle de Versões

Dos Sistemas de Gestão de Componentes analisados, o ***ComponentSource*** e ***Sensedia*** foram os únicos que possuem sistema de controle de versões. Os ***Spars-J***, ***Agora*** e ***CodeBroker*** não possuem controle de versão.

Segue abaixo o Quadro 8, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Controle de Versões.

Quadro 8 Análise Comparativa em Relação ao Controle de Versões

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Controle de Versões	Não	Sim	Sim	Não	Não

5.2.8 Controle de Modificações

Os Sistemas de Gestão de Componentes ***ComponentSource***, ***Sensedia***, ***Spars-J*** e ***CodeBroker*** possuem o controle de modificações. As modificações ocorridas nestes Sistemas de Gestão de Componentes são realizadas pelo administrador do sistema, cabendo ao usuário somente a solicitação da mudança requerida.

No Sistema de Gestão de Componentes ***Agora***, não há um controle de modificações.

Segue abaixo o Quadro 9, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Controle de Modificações.

Quadro 9 Análise Comparativa em Relação ao Controle de Modificações

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Controle de modificações	Sim	Sim	Sim	Não	Sim

5.2.9 Notificação de Mudanças

O ***ComponentSource*** e ***Sensedia*** possuem notificação de mudanças. As modificações ocorridas são informadas ao usuário através de *e-mail* (funcionalidade solicitada pelo usuário) e pela pagina principal do Sistema de Gestão de Componentes.

Os demais repositórios analisados não possuem notificação de mudanças.

Segue abaixo a Tabela 10, mostrando a comparação dos Sistemas de Gestão de Componentes em relação a Notificação de Mudanças.

Quadro 10 Análise Comparativa em Relação a Notificação de Mudanças

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Notificação de Mudanças	Não	Sim	Sim	Não	Não

5.2.10 Modelo de Negócio

Os Sistemas de Gestão de Componentes ***Spars-J***, ***Agora*** e ***CodeBroker*** são gratuitos.

O ***Sensedia*** é um Sistema de Gestão de Componentes pago, mas, após adquirido o Sistema, o resgate por componentes é feito de forma gratuita.

O ***ComponentSource*** é um Sistema de Gestão de Componentes no qual o acesso e o cadastro no *site* são gratuitos, mas a aquisição de componentes é paga e para cada componente é atribuído um valor que varia de acordo com a licença adquirida pelo usuário.

Segue abaixo o Quadro 11, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Modelo de Negócio.

Quadro 11 Análise Comparativa em Relação ao Modelo de Negócio

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Modelo de Negócio	Gratuito	Paga	Paga	Gratuito	Gratuito
Resgate do Componente	Gratuito	Gratuito	Paga	Gratuito	Gratuito

5.2.11 Tipo de Componentes

Os Sistemas de Gestão de Componentes ***Agora***, ***Spars-J*** e ***CodeBroker*** são exclusivos da plataforma Java, havendo apenas componente Java em seu repositório. O ***Sensedia*** suporta componentes de todos os tipos de plataformas, dependendo da necessidade do cliente. O ***ComponentSource*** trabalha com componentes das seguintes plataformas: .NET, Java, ActiveX/COM, JavaScript/AJAX, Flash/Flex, C++/MFC, DLL e VCL.

Segue abaixo o Quadro 12, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Tipo de Componente.

Quadro 12 Análise Comparativa em Relação ao Tipo de Componente.

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Tipos de Componentes	Somente Java	Qualquer tipo	.Net, Java, ActiveX/COM, JavaScript/ AJAX, Flash/Flex, C++/MFC, DLL e VCL	Somente Java	Somente Java

5.2.12 Tipo de Suporte

Dos Sistemas de Gestão de Componentes analisados, ***ComponentSource***, ***Spars-J***, ***Sensedia*** e ***Agora*** são acessados apenas pela Web, o ***CodeBroker*** pode ser acessado somente localmente.

Segue abaixo o Quadro 13, mostrando a comparação dos Sistemas de Gestão de Componentes em relação ao Tipo de Suporte.

Quadro 13 Análise Comparativa em Relação ao Tipo de Suporte.

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Tipo de Suporte	Somente Web	Somente Web	Somente Web	Somente Web	Somente Local

5.2.13 Resultados Obtidos

Após análise realizada entre os Sistemas de Gestão de Componentes foi possível gerar o Quadro 14, na qual reúne as principais características consideradas para o desenvolvimento deste trabalho.

Quadro 14 Tabela Geral de Comparações entre Características dos SGCs.

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Identificação	Palavra-chave	Palavra-chave	Palavra-chave	Palavra-chave	Palavra-chave
Permite inserção	Sim	Sim	Sim	Sim	Sim
Permite explorar o catálogo de componentes	Sim	Sim	Sim	Sim	Sim
Recuperação	Sim	Sim	Sim	Sim	Sim
Histórico	Não	Modificação, Inserção, Remoção	Somente de inserção	Não	Não
Controle de acesso	Sim	Sim	Sim	Não	Sim
Controle de versões	Não	Sim	Sim	Não	Não
Controle de modificação	Sim	Sim	Sim	Não	Sim

Continuação Quadro 14.

	SPARS-J	SENSEDIA	COMPONENTE SOURCE	AGORA	CODEBROKER
Notificação de Mudanças	Não	Sim	Sim	Não	Não
Modelo de negócio	Gratuito	Paga	Paga	Gratuito	Gratuito
Tipo de componentes	Somente Java	Qualquer tipo	.Net, Java, ActiveX/COM, JavaScript/ AJAX, Flash/Flex, C++/MFC, DLL e VCL	Somente Java	Somente Java
Tipo de Suporte	Somente Web	Somente Web	Somente Web	Somente Web	Somente Local

Analisando a tabela acima é possível concluir que cada SGC possui um perfil para um determinado tipo de usuário. Um exemplo são os SGCs ***ComponentSource*** e ***Sensedia*** que são voltados para usuário nos quais dispõem de dinheiro, pois ambos possuem uma política paga.

Por outro lado os SGCs ***Spars-J***, ***Agora*** e ***CodeBroker***, são voltados para usuário que pretendem utilizar o Desenvolvimento Baseado em Componentes (DBC), mas não disponibilizam de verbas, pois a política de uso do SGC é gratuita.

Os SGCs ***Spars-J***, ***Agora***, ***CodeBroker*** trabalha com componentes desenvolvidos em Java, vide critério "Tipo de Componente" no quadro de comparações. O ***ComponentSource*** trabalha com componentes desenvolvidos com .Net, Java, ActiveX/COM, JavaScript/ AJAX, Flash/Flex, C++/MFC, DLL e VCL. O ***Sensedia***, o mais flexível, trabalha com qualquer linguagem.

Para usuários que visam velocidade de desenvolvimento, o ***CodeBroker*** é uma boa opção. Nele, o usuário ganha tempo com as consultas, uma vez que o SGC é integrado ao ambiente de programação Emacs. Na medida em que o usuário desenvolve e adiciona comentários em seu código, o sistema

automaticamente busca e lista os componentes que tem maiores chances de serem reutilizados de acordo com o trecho de código digitado pelo programador.

O SGC ***Agora*** é uma boa opção para usuários que dispõem de tempo para a seleção de componentes, por ser vinculado ao sistema de busca AltaVista, as consultas são realizadas em toda a rede, gerando um grande numero de componentes para uma determinada palavra-chave.

A escolha final do SGC depende das características do projeto e cabe ao Engenheiro de Software e Gerente de Projeto escolherem a ferramenta mais adequada.

6 CONCLUSÃO

Este trabalho de pesquisa objetivou coletar as principais características dos Sistemas de Gestão de Componentes (SGC) e compará-las, o qual foi desenvolvido em um período de 1 ano no DCC (Departamento de Ciência da Computação) da UFLA (Universidade Federal de Lavras) como Trabalho de Conclusão de Curso (TCC).

Após a análise realizada, pode-se concluir que os Sistemas de Gestão de Componentes analisados, cujo Modelo de Negócio é gratuito, são voltados para a tecnologia Java, havendo falta de sistemas similares para outras tecnologias. Conclui-se também que os Sistemas de Gestão de Componentes gratuitos não possuem características como Histórico, Controle de Versão e Notificação de Mudanças.

Além disso, os SGCs pagos apresentaram melhores resultados em relação aos critérios estabelecidos. Destaca-se o ***Sensedia*** que além de permitir o gerenciamento de qualquer tipo de componente, foi o que atendeu os critérios de forma mais completa. Apesar do ***ComponentSource*** ter uma avaliação favorável, ele cria históricos apenas das inserções, não armazenando informações sobre modificações e remoções.

6.1 Principais Contribuições

O quadro comparativo apresentado e a descrição dos critérios em relação aos componentes analisados, auxiliam o Engenheiro de software a escolher o melhor SGC para seu projeto.

Alem disso, o presente trabalho contribui como material de pesquisa para trabalhos relacionados em SGCs.

6.2 Trabalhos futuros

Planeja-se, futuramente, reaplicar o trabalho dando mais flexibilidade para notas quantitativas em vez de Sim/Não. Sugere-se a aplicação da escala Likert, utilizando opções como “Não Atende”, “Atende Pouco”, “Médio/Regular”, “Atende Bastante” e “Atende Plenamente”. Tais valores são convertidos para números, possibilitando uma classificação entre os SGCs.

Replicar o trabalho com novos critérios como: Maturidade da Solução (numero de ano que ela existe), Maturidade da Empresa (tempo de existência da empresa), Garantia de Documentação do Componente, dentre outros.

Replicar o trabalho analisando em SGCs de tecnologias específicas como Java, .NET, FLEX, com o objetivo de analisar qual o melhor SGC para uma determinada tecnologia.

Por fim, propor um modelo de Sistema de Gestão de Componentes de Software sem as deficiências ou aproveitar um projeto de software livre de código-aberto e gratuito para melhorá-lo, atendendo aos critérios estabelecidos.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES A.; LUCCA J. E.; CARNEIRO A. C.; FERREIRA C.; VEIGA R.; MINODA R.; TACCA T.; REZENDE A.; WIT A. **Perspectivas de desenvolvimento e uso de componentes na Indústria Brasileira de Software e Serviços**, 2007.

BASS, L.; BUHMAN, C.; DORDA, S.; LONG, F; ROBERT, J.; SEACORD, R.; WALLNAU, K. **Market Assessment of Component-Based Software Engineering**. SEI, Technical Report CMU/SEI-2001-TN-007, 2000.

BRADSHAW, T.K.; DUMAIS, S.T. **A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge**, Psychological Review, 104(2), 211-240, 1997.

BROWN, A.W., SHORT, K. **Components and Objects: The Foundation of Component-Based Development**. International Symposiumn Assessment of Software Tools and Technologies (sast 97): IEEE Computer Society Press.; Pags. 112-121, 1997

COMPONENTSOURCE. Disponivel em: <www.componentsource.com>. acesso em 06/05/2010.

COMPONENT RANK. **Component Rank: Relative Significance Rank for Software Component Search**. Sem data (S.d).

CONSTANTOPOULOS, P.; DOERR, M.;VASSILIOU, Y. **Repositories for Software Reuse: The Software Information Base**. In: WORKINGCONFERENCE ON INFORMATION SYSTEM DEVELOPMENT PROCESS, 1993,Como, Italy. Proceedings… Como, Italy: North-Holland. p. 285-307, 1993.

CRNKOVIC, I. **Component-Based Software Engineering–New Challenges in Software Development**. Information Technology Interfaces, pp. 127-133, Croatia, 2003

D'SOUZA, D. F. e WILLS, C. A. **Objects, Components, and Frameworks with UML**. The Catalysis Approach. Addison-Wesley, 1999.

EZRAN, M. **Practical Software Reuse: The Essencial guide**.185 p, 1999.

FRAKES, W.; KANG, K. **Software Reuse Research: Status and Future.** IEEE Transactions on Software Engineering, Vol.31, Nº 7, July, 2005.

GIL, A. C. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 1991.

GRUNDY, J. **Storage and Retrieval of Software Components Using Aspects.** In: AUSTRALASIAN COMPUTER SCIENCE CONFERENCE , 2000.

GUERRA A. C. e ALVES A. M. **Aquisição de Produtos e Serviços de Software**. Ed. Campus, 2004

GUO J., "**Toward Automated Retrieval for a Software Component Repository**", IEEE Conference and Workshop on Engineering of Computer-Based Systems, 1999

HENNINGER, S. **An Evolutionary Approach to Constructing Effective Software Reuse Repositories**. In : ACM Transactions on Software Engineering and Methodology, 1997.

JUNG, C. F. **Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos**. Rio de Janeiro/RJ: Axcel Books do Brasil Editora, 2004.

KONTIO J.; CALDEIRA G.; BASILI V.R. **Defining Factors, Goals and Criteria for Reusable Component Evaluation**, 1996.

KONTIO, J**. A Case Study in Applying a Systematic Method for COTS Selection.** 18th International Conference on Software Engineering in Berlin, Germany, 1996.

KOTONYA, G.; HUTCHINSON, J.; SAWYER, P.; WALTER, O.; JOAN, C. **COTS Component-Based System Development: Processes and Problems**, 2002.

LAKATOS, E. V.; MARCONI, M. de A. **Metodologia científica**. São Paulo/SP: Editora Atlas, 1988.

LANDAUER, T. K.; DUMAIS, S. T., "**A Solution to Plato's Problem: The Latent Semantic Analysis Theory of Acquisition, Induction and Representation of Knowledge**," Psychological Review, 104(2), 211-240, 1997.

OMG. **Reusable Asset Specification, version 2.2**. Disponível online <http://www.omg.org/technology/documents/formal/ras.htm>. Ultima Versão 11/02/2005. Último acesso em 15 de maio de 2010.

PETERS, J. F. **Engenharia de Software**,Rio de Janeiro,Ed. Campus, 2001.

PIMENTEL, M. G.C.; TEIXEIRA, C. A. C.; SANTANCHE, A.. **XML: Explorando suas Aplicações na Web**. Congresso da Sociedade Brasileira de Computação, Curitiba 2000.

PITTS-MOULTIS, N., KIRK, C. **XML Black Book - Solução e Poder**. Makron Books, 627 p, 2000.

RESENDE, A. R. M. de L. **Um Modelo de Processo para Seleção de Componentes de Software** / Ana Rubélia Mendes de Lima Resende. São José dos Campos, 2006 215f, 2006.

ROSSI, A. C. **Representação de software na FARCSOFT: Ferramenta de apoio à reutilização de componentes de software**. Dissertação de mestrado. Poli/USP, 2004.

SAMETINGER, J. Software Engineering with Reusable Components. 1. ed. Berlin, Germany, Springer-Verlag, 1997.

SEACORD, R. C. **Agora: A Search Engine for Software Components**. Technicalreport, Software Engineering Institute (SEI), 1999.

_____. **Software Engineering Component Repositories**. Technicalreport, Software Engineering Institute (SEI), 1999.

SILVA, E. L.; MENEZES, E. M. **Metodologia da pesquisa e elaboração de dissertação**. Florianópolis, 2000.

SZYPERSKI, C. "**Component Technology – What, Where and How?**". In: Proceedings of the 25th International Conference on Software Engineering (ICSE.03), 2003.

W3 CONSORTIUM : Grupo Baseado no MIT – ISO – *International Standard Organization*. Disponível em: <http://ww.iso.org>, acesso em 29/05/2010

YOKOMORI, R; ISHIO, T; YAMAMOTO, T; MATSUSHITA, M; KUSUMOTO, S.; INOUE, K. **Java Program Analysis Projects in Osaka University: Aspect-Based Slicing System ADAS and Ranked-Component Search System SPARS-J**, Sem Data (s.d)

YUNWEN, Y.; FISCHER, G; REEVES, B. **Integrating Active Information Delivery and Reuse Repository Systems**, 2000.

ZAMBALDE, A. L.; PÁDUA, C. I. P. S.; ALVES, R. M. **O Documento Científico em Ciência da Computação e Sistemas de Informação**. 1 ed, 2008. Disponível em: <www.dcc.ufla.br/~zambalde/aulas/Apostila_PDF.pdf>, acesso em Abril de 2010.

Zaremski, A. M.; J. M. WING, "**Signature Matching: A Tool for Using Software Libraries**," ACM Trans. Soft. Eng. Meth., 4(2), 146-170, 1995.