Pós-Graduação em Ciência da Computação

"SPECIFICATION, DESIGN AND IMPLEMENTATION OF A REUSE REPOSITORY"

Por

Vanilson André de Arruda Burégio

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, SETEMBRO/2006

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

VANILSON ANDRÉ DE ARRUDA BURÉGIO

“Specification, Design and Implementation of a Reuse Repository"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR(A): Silvio Romero de Lemos Meira

RECIFE, SETEMBRO/2006

Burégio, Vanilson André de Arruda
Specification, design and implementation of a reuse repository / Vanilson André de Arruda Burégio. - Recife : O Autor, 2006.
ix, 131 p. : il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2006.

Inclui bibliografia e apêndices.

1. Engenharia de software. 2. Reuso de software. 3. Componentes de software. I. Título.

005.1 CDD (22.ed.) MEI2008-068

gradecimentos

Primeiramente, quero aqui registrar meus agradecimentos a Deus, que tanto me deu força e coragem durante todo esse percurso e me concedeu a oportunidade de realizar este trabalho. Sem Ele nada disso seria possível.

Em seguida, agradeço ao meu orientador Silvio Romero de Lemos Meira, pela orientação, pelo incentivo e, principalmente, por me proporcionar a oportunidade de desenvolver um trabalho de mestrado que pudesse ser aplicado em um projeto real de desenvolvimento de software.

Ao grande pesquisador e amigo Eduardo Almeida, por sempre ter acreditado no sucesso deste trabalho e ter me dado total apoio nos momentos de extrema dificuldade; pelas críticas, sugestões, revisões e por todo incentivo e disponibilidade em sempre querer ajudar.

Obrigado a todos os membros do grupo RiSE, pelas discussões, sugestões e críticas fornecidas, especialmente a Alexandre Martins, Eduardo Cruz, Liana Lisboa e Daniel Lucrédio pela prontidão prestada nos momentos difíceis do trabalho.

Ao Centro de Estudos e Sistemas Avançados do Recife – CESAR, pelo apoio técnico, pessoal e financeiro. Em especial aos membros do projeto GComp, principalmente, a Juliana Dantas que sempre me incentivou a alcançar essa conquista tão almejada. Gostaria de agradecer também a colaboração de Luiz Eugênio e os demais participantes envolvidos no processo de avaliação de arquitetura.

Agradeço à SEFAZ, na pessoa de Nevton Borba, por ter me prestado total apoio em momentos de difícil conciliação entre o trabalho e a vida acadêmica.

Obrigado aos meus pais e a toda minha família pelo grande incentivo prestado no decorrer de todas as atividades envolvidas na confecção deste trabalho. Em especial, dedico este trabalho a minha avó Creuza que, onde quer que esteja, tenho certeza de que continua torcendo e vibrando pelas minhas conquistas.

esumo

A disciplina de Reuso de Software tem crescido em importância, tornando-se uma ferramenta estratégica para empresas que almejam um aumento de produtividade, a obtenção de baixos custos e a alta qualidade dos seus produtos.

Porém, antes de obtermos as vantagens inerentes ao reuso, é preciso termos mecanismos hábeis a fim de facilitar o armazenamento, a busca, a recuperação e o gerenciamento dos artefatos reusáveis. Nesse contexto, encaixa-se a idéia de repositórios de reuso. Um repositório de reuso pode ser entendido como uma base preparada para o armazenamento e a recuperação de componentes. O mesmo pode ser visto também, como um grande facilitador, que atua como suporte aos engenheiros de software e outros usuários no processo de desenvolvimento de software *para* e *com* reuso.

Na literatura, existem diversos trabalhos que exploram repositórios de reuso, porém, o foco desses trabalhos está, quase sempre, voltado a questões de busca e recuperação de componentes e, muitas vezes, aspectos importantes de repositórios de reuso não são explorados adequadamente, como, por exemplo, o uso destes como ferramenta para auxiliar gerentes no monitoramento e controle do reuso em uma organização.

Por outro lado, algumas questões levantadas por empresas que desejam construir um repositório de reuso continuam mal respondidas. Tais questões geralmente incluem: Que papéis um repositório deve desempenhar no contexto de reutilização? Quais são os principais requisitos de um repositório de reuso? Quais as alternativas práticas existentes? Como um repositório de reuso pode ser projetado?

Motivado por essas questões, esta dissertação apresenta a especificação, o projeto e a implementação de um repositório de reuso baseado na análise das soluções existentes e em uma experiência prática de construção de um ambiente de reuso para fábricas de software. Adicionalmente, são discutidos os resultados obtidos, os problemas encontrados, e as direções futuras para pesquisa e o desenvolvimento.

Palavras-chave: Repositório de Reuso, Componentes de Software, Reuso de Software

bstract

The software reuse subject has increased in importance, becoming a strategic tool for enterprises that aspire to a productivity boost, low costs and high quality of their products.

Nevertheless, before obtaining the advantages relevant to reuse, it is necessary that we have useful mechanisms in a way to make storage, search, recovery and managing of reusable artifacts easier. In this context is where the idea of reuse repositories perfectly fits. A reuse repository can be understood as a prepared base for the storage and recovery of components. It can also be seen as a big facilitator which acts as a support to software engineers as well as other users in software development process for and with reuse.

In the literature, several works that explore reuse repositories are available. However, the focus of these works is quite often related to matters of search and recovery of components and, time and again, important aspects of reuse repositories are not explored properly, for instance, their use to aid managers in the monitoring and reuse control in an organization.

On the other hand, some questions brought up by companies that aim at building a reuse repository remain unanswered. Such questions usually include: which roles should a repository play in a reutilization context? What are the main requisites of a reuse repository? What are the existing practical alternatives? How can a reuse repository be designed?

Fostered by these questions, this dissertation presents the specification, the project and implementation of a reuse repository based on the analysis of existing solutions as well as on a building practical experience of a reuse environment for software factories. In addition, the obtained results, setbacks found as well as future directions for research and development will be presented.

Keywords: Reuse Repository, Software Components, Software Reuse

Conteúdo

Lista de Figuras

Lista de Tabelas

Introdução

1.1 Motivação

Existe um consenso que empresas de software estão sempre buscando produzir produtos com um menor custo e tempo e com uma maior qualidade. Neste sentido, o reuso de software tem crescido em importância e se tornado uma abordagem indispensável em busca desses objetivos. A idéia básica de reuso é construir software através da utilização de artefatos ou conhecimento já existentes, ao invés de iniciar o desenvolvimento do zero, sem nenhum reaproveitamento [Krueger, 1992].

Algumas experiências práticas na indústria [Basili, 1991] [Bauer, 1993] [Griss, 1994] [Joos, 1994] [Griss, 1995] têm comprovado que o reuso de software aumenta a produtividade e ajuda a obter baixos custos e alta qualidade durante o ciclo de desenvolvimento de software como um todo.

Porém, reuso não é uma tarefa de desenvolvimento simples e o sucesso de um programa de reuso, muitas vezes, depende da aplicação efetiva de *aspectos técnicos* e *não técnicos*. Os *aspectos não-técnicos* incluem questões como: educação, treinamento, incentivos e gerenciamento organizacional. Por outro lado, os *aspectos técnicos* compreendem a utilização de métodos, processos, ferramentas e ambientes que forneçam suporte ao desenvolvimento e posterior reuso de software.

Geralmente, a falta de conhecimento sobre a disponibilidade de artefatos de software que podem ser reutilizados, aliada ao custo de se encontrar tais artefatos são as principais causas para que desenvolvedores não tentem reusar software [Frakes & Fox, 1995]. Como mostra a Figura 1.1, o desenvolvedor enfrenta o

problema de localizar, entender e integrar software reutilizável em um novo software a ser criado, pois, o conhecimento disponível normalmente é desestruturado e está espalhado nas diversas *ilhas de informação*. Na prática, em uma empresa, estas ilhas podem representar departamentos, equipes de projetos e outras unidades organizacionais que geram software e/ou conhecimento não compartilhado.

Por esta razão, o problema de localizar software reutilizável é um aspecto fundamental para um programa de reuso efetivo [Almeida et al., 2004][Mascena, 2006][Vanderlei, 2006].

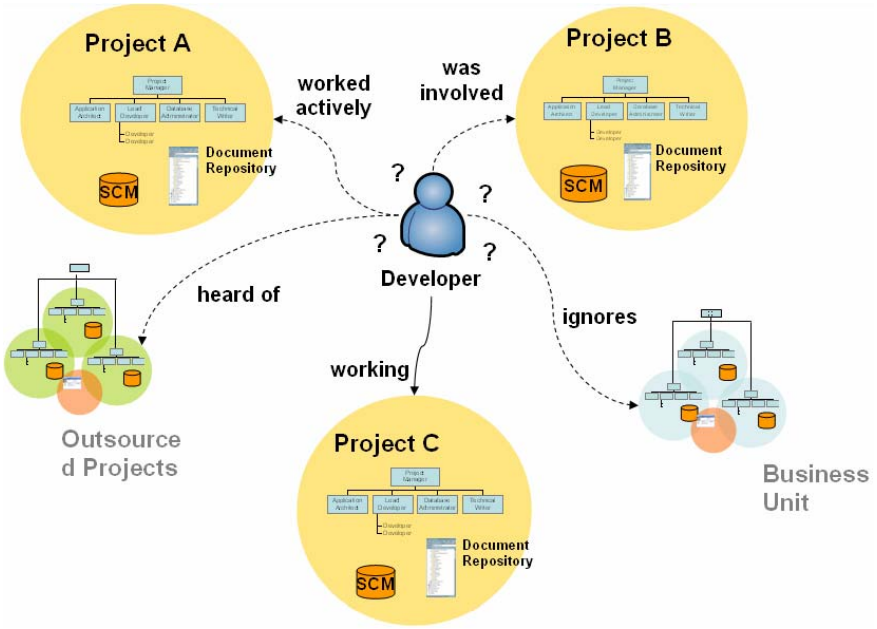


Figura 1.1 - Ilhas de informação

Desta forma, antes dos desenvolvedores obterem as vantagens inerentes ao reuso, eles devem possuir mecanismos hábeis a fim de facilitar o armazenamento, a busca, a recuperação e a distribuição dos artefatos reutilizáveis. É neste contexto que se encaixa a idéia de *sistemas de repositório*. Um sistema de repositório pode ser entendido como um sistema de informação que auxilia os programadores na localização dos artefatos reusáveis [Ye, 2002].

Guo & Luqi [Guo & Luqi, 2000] se referem a sistemas de repositórios (*bibliotecas de reuso*) como sendo organizações de pessoas, procedimentos, ferramentas e componentes direcionados a facilitar que o reuso de componentes de software alcance objetivos específicos de produtividade e custo-benefício.

De acordo com Poulin [Poulin, 1995], *Bibliotecas de Software Reutilizáveis* formam, quase sempre, o **centro de uma estratégia de reuso** organizacional. De fato, Frakes [Frakes, 1994] afirma que biblioteca de software é um **fator crítico** para se implementar um programa de reuso sistemático e ele complementa sua afirmação dizendo:

"*Uma vez que uma organização adquire ativos reutilizáveis, ela tem que possuir uma maneira de armazená-los, buscá-los e recuperá-los – uma biblioteca de reuso.*" (p. 19)

Isso reforça a idéia de que repositórios representam um recurso precioso para os desenvolvedores, além de formarem a infra-estrutura básica de suporte ao processo de reutilização de software [Mili et al., 1997].

Contudo, apesar de sua importância, o uso de repositórios para suportar um processo de reuso tem muitas vezes falhado na prática [Frakes, 1995] [Morisio, 2002]. Uma das razões para estas falhas é que, quase sempre, um repositório é construído apenas sob a perspectiva de desenvolvedores. Ou seja, os repositórios são tidos como simples mecanismos de busca e recuperação de componentes e importantes aspectos, como, por exemplo, o uso destes como ferramenta para auxiliar gerentes no **monitoramento e controle do reuso** em uma organização não são tratados.

Além disso, segundo Wohlin [Wohlin, 1998], para que os componentes possam ser recuperados de uma infra-estrutura, como um repositório, eles devem possuir um certo grau de qualidade, definindo eficientemente o nível de confiabilidade que pode ser esperado destes componentes, antes que eles sejam reusados no desenvolvimento de uma aplicação. Assim, é extremamente importante **assegurar a qualidade** dos componentes que estão sendo armazenados no repositório através do suporte a um processo de certificação de componentes adequado [Álvaro, 2006]. Não obstante, esse aspecto raramente é tratado nas soluções existentes, conforme será visto no próximo Capítulo.

De fato, na literatura, existem diversos trabalhos que exploram a utilização de repositórios no contexto de reuso de software, porém, o foco desses trabalhos está, quase sempre, voltado a questões de busca e recuperação de componentes e, na maioria das vezes, os aspectos organizacionais e de qualidade não são explorados adequadamente.

Por outro lado, durante a última década, várias organizações comerciais e industriais têm desenvolvido diversas soluções que permitem o acesso a componentes de software e os seus recursos relacionados [Pan et al., 2004]. Na prática, as soluções adotadas por empresas variam desde o uso de ferramentas específicas de reuso (projetadas com o objetivo principal de promover o reuso de software) a ferramentas de uso geral que, muitas vezes, correspondem a ferramentas tradicionais de desenvolvimento de software adaptadas para servirem como repositórios de componentes.

Desta forma, algumas questões levantadas por empresas que desejam construir ou adotar um repositório para apoiar suas atividades de reuso continuam mal respondidas. Tais questões, geralmente incluem: *Que papéis um repositório deve desempenhar no contexto de reutilização? Quais são os principais requisitos de um repositório de reuso? Quais as alternativas práticas existentes? Como um repositório de reuso pode ser projetado e implementado?*

1.2 Estabelecimento do problema

Motivado pelas questões apresentadas na Seção anterior, o principal objetivo deste trabalho é, em linhas gerais:

Especificar, projetar e implementar um repositório de reuso corporativo que possa auxiliar tanto desenvolvedores na produção e consumo de ativos de software reutilizáveis, bem como prover meios para monitorar as atividades de reuso e permitir a garantia de qualidade dos ativos disponibilizados.

1.3 Visão geral da solução proposta

Com o objetivo de alcançar o objetivo deste trabalho, estabelecido na Seção anterior, foi realizado um estudo das soluções de repositório de reuso existentes tanto na academia, quanto na indústria. Este estudo teve como finalidade identificar o tipo de solução mais adequada para desempenhar os papéis do repositório proposto. A partir desse estudo foram definidos os requisitos e, em seguida, a arquitetura da solução, que passou por um processo formal de avaliação de arquitetura [Clements et al., 2002a].

Depois de especificado e projetado, o repositório proposto foi implementado dentro de um projeto de desenvolvimento real que gerou um produto comercial.

As próximas Seções descrevem o contexto deste trabalho e apresenta uma visão geral da arquitetura da solução.

Contexto

Este trabalho é parte de uma grande iniciativa de reuso promovida pelo grupo de pesquisa *Reuse in Software Engineering* (RiSE[1]) [Almeida et al., 2004]. A Figura 1.2 mostra o *framework* global do projeto RiSE.

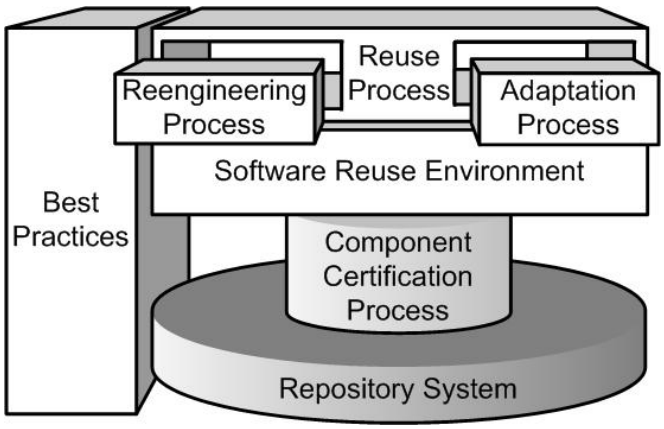


Figura 1.2 – Framework do projeto RiSE

O projeto RiSE engloba diversos aspectos de reuso não incluídos no escopo deste trabalho, tais como, processo de reuso [Almeida et al., 2005b], certificação de componentes [Alvaro et al., 2005b] e ambientes de reuso de software [Garcia et al.,

[1] http://www.rise.com.br

2006b]. Outras ferramentas e ambientes propostos pelo grupo incluem, entre outras, o engenho de busca Maracatu [Garcia et al., 2006], a ferramenta de análise de domínio *ToolDay* [Lisboa, 2006], o ambiente de reuso integrado baseado em métricas *ADMIRE* [Mascena, 2006] e o engenho de busca de artefatos com suporte a *folksonomia* [Vanderlei, 2006]. Estes esforços são coordenados e farão parte de uma solução completa de reuso a ser aplicada em contextos corporativos.

No caso particular deste trabalho, uma implementação baseada na solução proposta por esta dissertação foi desenvolvida em conjunto com o Centro de Estudos e Sistemas Avançados do Recife (C.E.S.A.R.) e teve o apoio (financiamento) do governo, representado pela FINEP, a qual publicou (no DOC de 31/08/2004) uma seleção pública de propostas para apoio à cooperação tecnológica entre o setor produtivo e instituições científicas e tecnológicas (CHAMADA PÚBLICA MCT/FINEP/Ação Transversal-Cooperação ICTs-Empresas-02/2004 - Referência FINEP n.º 2176/04).

O desenvolvimento do repositório proposto fez parte da principal meta do projeto industrial submetido à FINEP. Tal projeto envolveu também, a criação de processos, métodos e ferramentas para suportar as atividades de reuso de software em empresas.

Visão geral da arquitetura

A arquitetura do repositório proposto consiste em um conjunto de módulos e componentes que trabalham em conjunto para prover as funcionalidades necessárias de um repositório de reuso efetivo. A Figura 1.3 mostra a arquitetura proposta.

Os três principais módulos de negócio da arquitetura são: *Produção, Gerenciamento* e *Consumo*. Estes módulos provêem as funcionalidades essenciais utilizadas pelos diferentes tipos de usuários do sistema.

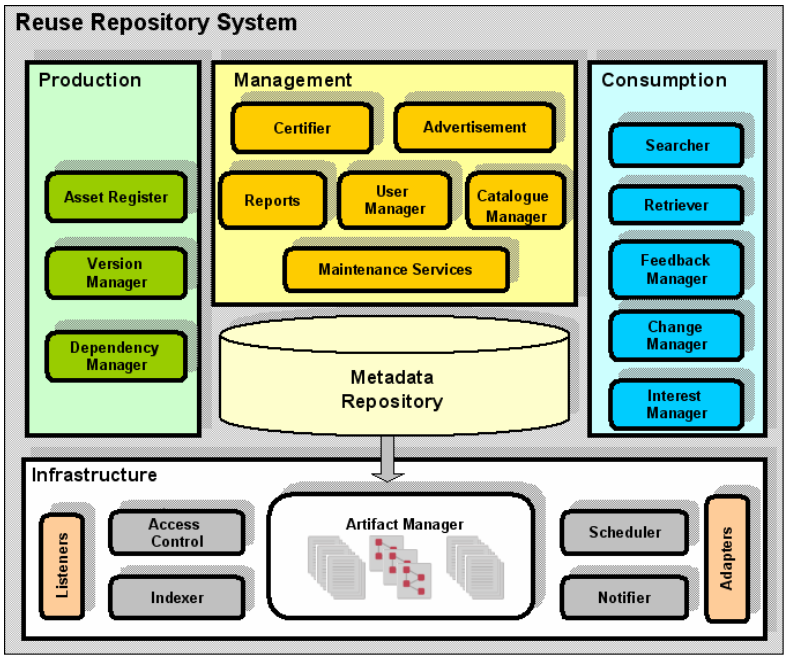


Figura 1.3 – Arquitetura da solução proposta

O módulo de *Produção* oferece os serviços para os desenvolvedores que produzem os ativos reutilizáveis. O módulo de *Gerenciamento* concentra os serviços que apóiam a monitoração das atividades de reuso na organização e a garantia de qualidade dos ativos disponibilizados. Por fim, o módulo de *Consumo* auxilia os usuários na busca e recuperação dos ativos, além de oferecer outros serviços úteis para os usuários que reusam os ativos do repositório.

O elemento central da arquitetura, que é utilizado pelos três módulos de negócio, é o *Metadata Repository*, que funciona como uma base de dados compartilhada para armazenar e recuperar os metadados dos ativos disponibilizados. Ele faz parte do módulo de infra-estrutura (*Infrastructure*) que, além dele, é formado por um conjunto compartilhado de componentes e serviços que podem ser utilizados por todos os outros módulos do repositório. Os detalhes da arquitetura serão descritos na Seção 3.2.

1.4 Escopo negativo

Como a solução proposta por esta dissertação faz parte de um contexto mais amplo, um conjunto de aspectos relacionados não será tratado no escopo deste trabalho.

Adicionalmente, os requisitos tratados na solução não formam um conjunto completo e fechado de funcionalidades que devem sempre estar presentes em qualquer repositório de reuso, pois isso dependerá das reais necessidades e do contexto de cada empresa onde o mesmo será utilizado. Contudo, acredita-se que os requisitos identificados podem servir como base para a construção de repositórios que suportem um processo de reuso em organizações.

Os seguintes aspectos não fazem parte do escopo deste trabalho:

- **Processo de reuso:** embora seja feita uma breve explicação de onde um repositório de reuso se encontra dentro do contexto de reutilização, e os papéis que o mesmo deve desempenhar, não serão tratadas, com detalhe, todas as fases de um processo de reuso a ser suportado pelo repositório proposto;
- **Funcionalidades:** algumas das funcionalidades previstas na arquitetura, tais como, um *extrator automático de ativos* [Mendes, 2006], não foram implementadas na versão inicial do repositório. Estas funcionalidades, embora importantes, consistem em ferramentas de suporte para as funcionalidades principais do repositório e podem ser gradualmente implementadas em versões futuras do repositório;
- **Atributos de qualidade:** as funcionalidades do repositório (requisitos funcionais) foram os principais aspectos tratados na especificação de requisitos. Embora considerados na implementação e no processo de avaliação de arquitetura, alguns atributos de qualidade, como, por exemplo, distribuição, performance e escalabilidade não foram detalhados;
- **Busca:** as questões relacionadas com busca no repositório são tratadas de uma maneira geral sem detalhes a respeito dos algoritmos de indexação e classificação. Além disso, aspectos relacionados a implementação de *busca ativa* no repositório [Ye, 2001] [Mascena, 2006] também não serão tratados; e
- **Modelo de Negócio:** Aspectos relacionados ao suporte a modelos de negócio que permitem a negociação de componentes através de repositórios não serão tratados neste trabalho.

1.5 Principais Contribuições

Como resultado do trabalho apresentado nesta dissertação, as seguintes contribuições podem ser destacadas:

- **Definição de um modelo de ativo**: baseado na análise de como componentes de software são empacotados e caracterizados em algumas abordagens e soluções existentes, um modelo de ativo foi definido com o objetivo de possuir as informações adicionais manipuladas pelo repositório proposto;
- **Papéis de um repositório de reuso:** os principais papéis que um repositório deve desempenhar em um contexto de reutilização de software foram propostos;
- **Taxonomia das soluções existentes:** através da análise das ferramentas existentes foi criada uma taxonomia que classifica os tipos de soluções que podem ser usadas como repositórios de reuso por empresas;
- **Requisitos de um repositório de reuso:** a partir da análise das soluções existentes e dos papéis que um repositório de reuso deve desempenhar, um conjunto de requisitos para um repositório de reuso efetivo foram especificados;
- **Arquitetura do repositório:** os detalhes de uma arquitetura extensível de um repositório de reuso voltado ao gerenciamento de ativos foi definida;
- **Produto comercial:** baseado na proposta desta dissertação um produto comercial foi implementado e está atualmente implantado em fábricas de software brasileiras;
- **Avaliação de arquitetura:** O desenvolvimento do produto comercial envolveu, de maneira inovadora, a aplicação de um método formal de avaliação de arquitetura, onde pode ser comprovada a utilidade dessa abordagem dentro de um contexto industrial.

Além das contribuições finais listadas acima, alguns resultados intermediários deste trabalho estão sendo reportados na literatura, como mostrado a seguir:

- Livro do grupo RiSE, **C.R.u.i.S.E.: Component Reuse in Software Engineering**, Recife, Brasil, (a aparecer) 2006;
- V. Burégio, E. Santos, E. Almeida, S. Meira, **Industrial Experiment in Software Architecture Evaluation,** WICSA'2007 - Working IEEE/IFIP Conference on Software Architecture, Mumbai, India, 2007 (em avaliação).
- V. Burégio, E. Almeida, D. Lucredio, S. Meira, **Specification, Design and Implementation of An Industrial Reuse Repository,** Journal Software Practice and Experience (SPE), 2006 (em preparação).
- V. Burégio, E. Almeida, S. Meira, **Industrial Experiences with Software Architecture Evaluation: The Case of the Reuse Tools,** Journal System and Software (JSS), 2006 (em preparação).

1.6 Organização da dissertação

O restante desta dissertação está organizado como se segue:

- **Capítulo 2**: apresenta a revisão bibliográfica sobre sistemas de repositórios aplicados ao contexto de reuso de software, com o objetivo de identificar as principais soluções existentes na academia e na indústria e analisar como as mesmas desempenham os papéis desejados de um repositório de reuso;
- **Capítulo 3**: descreve em detalhes os requisitos do repositório proposto, sua arquitetura e as tecnologias e *frameworks* adotados na sua implementação;
- **Capítulo 4**: apresenta alguns exemplos de uso da ferramenta desenvolvida, discute o processo de avaliação de arquitetura realizado durante o desenvolvimento da solução e apresenta algumas decisões técnicas realizadas durante o desenvolvimento;
- **Capítulo 5**: conclui esta dissertação através de um resumo das principais contribuições desta pesquisa e discussões a respeito de alguns trabalhos relacionados. Por fim, são também apresentadas algumas observações finais e direções para pesquisas futuras.

2 Repositórios de Reuso

Como visto no Capítulo anterior, acredita-se amplamente que reuso de software, um processo de criação de sistemas de software a partir de software existente, é um possível caminho para melhorar a produtividade e a qualidade do desenvolvimento de software. Porém, antes de desenvolver com reuso, os engenheiros de software devem ser capazes de eficiente e convenientemente adquirir os artefatos que desejam [Pan et al., 2004].

Neste contexto, vários trabalhos na academia e na indústria têm explorado o uso de sistemas de repositório como um meio de prover acesso aos artefatos disponíveis. Os sistemas de repositório tratados nestes trabalhos recebem variadas denominações, tais como: “repositório de componentes”, “bibliotecas de software”, “gerenciadores de artefatos reutilizáveis”, entre outros. Neste Capítulo será usado o termo “repositório de reuso”, genericamente, como uma denominação às diversas soluções que visam a aplicação de sistemas de repositórios no contexto de reutilização.

Deste modo, este capítulo apresenta uma visão geral sobre os diferentes tipos de repositórios de reuso e está organizado da seguinte forma: a **Seção 2.1** propõe os papéis que um repositório de reuso deve desempenhar. A **Seção 2.2** discute alguns trabalhos na academia que tratam questões relacionadas a repositórios de reuso. Na **Seção 2.3** as soluções práticas existentes são classificadas. A **Seção 2.4** apresenta uma comparação dessas soluções. A **Seção 2.5** examina os principais gerenciadores de componentes do mercado, e finalmente, a **Seção 2.6** resume a discussão a respeito de repositórios de reuso.

2.1 Papéis de um Repositório

Inicialmente, é importante entender onde um repositório aparece no contexto de um processo baseado em reuso e quais são os papéis que o mesmo deve desempenhar de acordo com os interesses das pessoas envolvidas em tal processo (*stakeholders*).

Em qualquer processo de desenvolvimento baseado em reuso, podem ser distinguidas, pelo menos, duas atividades: o desenvolvimento *para* reuso e o desenvolvimento *com* reuso. Um bom exemplo de uma técnica baseada em reuso, na qual estas atividades são bem definidas, é o Desenvolvimento Baseado em Componentes (DBC).

O DBC surgiu como uma nova perspectiva para o desenvolvimento de software, cujo objetivo é a construção de software através de componentes interoperáveis, reduzindo, desta forma, a complexidade do desenvolvimento, assim como os custos, através da utilização de componentes que, em princípio, seriam adequados para serem utilizados em outras aplicações [Sametinger, 1997]. Ele será utilizado aqui para ilustrar mais claramente os papéis que um repositório de reuso deve desempenhar.

Segundo Apperly [Apperly, 2001], no DBC tem-se o processo de *produção-gerência-consumo* onde as seguintes atividades são identificadas:

- **Produção**: focada na produção e disponibilização (publicação) de componentes, isto é, o desenvolvimento *para* reuso;
- **Gerência**: direcionada basicamente no monitoramento do reuso e no controle de qualidade dos componentes disponibilizados; e
- **Consumo**: concetrada na busca, recuperação e uso dos componentes disponíveis (desenvolvimento *com* reuso).

Em geral, as atividades de produção e consumo são executadas por desenvolvedores identificados como *produtores de componentes* e *consumidores*

de componentes, respectivamente; e as atividades de gerenciamento podem ser executadas, por exemplo, por gerentes de projeto e certificadores de componentes.

Portanto, neste contexto, há um requisito essencial para construir um repositório não só como uma base estática de componentes, mas também como uma ferramenta que suporte as atividades de publicação, consumo e gerência de tais componentes. A Figura 2.1 mostra onde um repositório de reuso aparece no processo de *produção-gerência-consumo*.

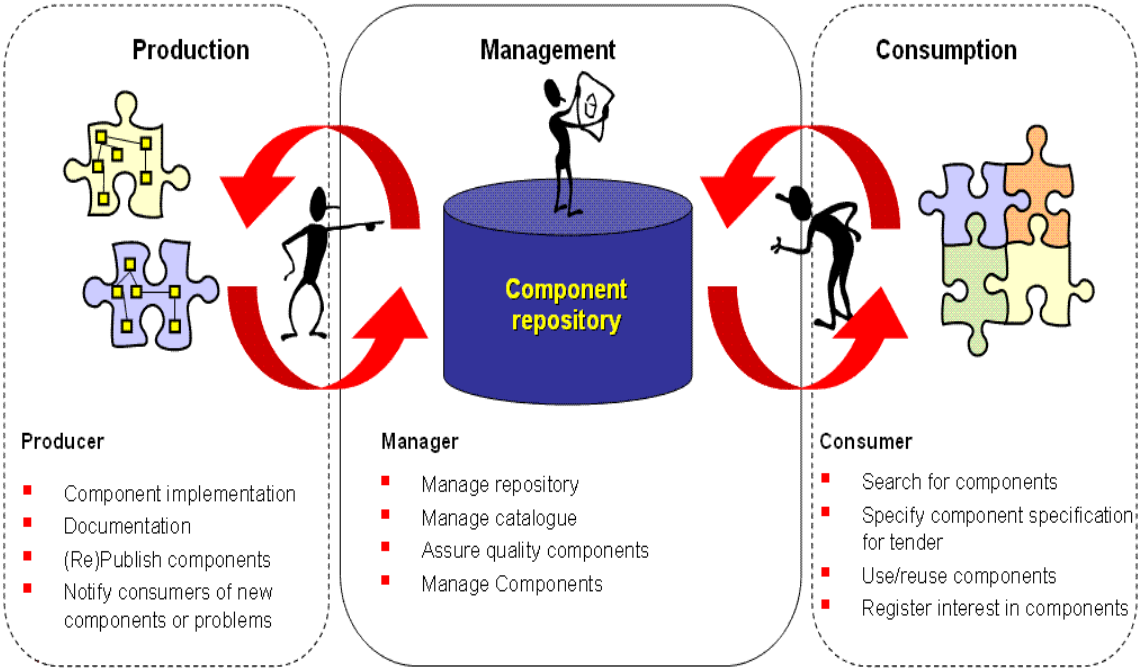


Figura 2.1 – Processo de produção-gerência-consumo

Como apresentado na Figura 2.1, o repositório deve conter serviços que satisfazem os interesses de três tipos de *stakeholders*: produtor, gerentes e consumidor. Baseado em tais interesses e com o objetivo de tornar os benefícios do reuso de software – produtividade, alta qualidade e baixos custos – mais facilmente alcançáveis, podem ser definidos alguns papéis que um repositório deve assumir dentro do contexto de reutilização. Para a definição desses papéis foram levados em consideração aspectos práticos em conjunto com o estado da arte na área.

É esperado que estes papéis possam desempenhar uma função relevante no suporte ao reuso sistemático de software. Os papéis propostos são:

2.1.1 Repositório como um veículo de comunicação entre os stakeholders

Este é um dos papéis essenciais de um repositório de reuso. Como apresentado na Figura 2.1, repositórios representam um canal de distribuição de artefatos, portanto, devem servir como um veículo de comunicação em duas direções: 1 - desenvolvimento *para* reuso (produção) e 2 - desenvolvimento *com* reuso (consumo). Este papel estabelece que um repositório deve ser visto não só como uma base estática de componentes, mas também como um mecanismo central que permita a interação entre os produtores e os consumidores de componentes.

Em [Apperly, 2001], é feita uma analogia entre repositório de componentes e o pátio de uma construção civil. Tal analogia reforça a idéia que um repositório pode desempenhar um papel chave quando componentes são negociados entre produtores, que estão interessados em publicar e/ou vender seus componentes, e consumidores, que se preocupam em buscar e recuperar componentes para a construção de novas aplicações.

2.1.2 Repositório como assistente no gerenciamento

Reuso de artefatos de software constituem oportunidades plausíveis para impulsionar a produtividade. Porém, junto com estas oportunidades surge a necessidade de monitorar o reuso no nível organizacional [Banker et al., 1993]. De fato, as atividades de reuso englobam o envolvimento de múltiplos projetos e

sistemas de aplicações. Para gerenciar tais atividades, duas tarefas fundamentais devem ser empregadas no nível organizacional: a coleta de métricas e o monitoramanto de software.

Neste sentido, repositórios podem ser usados por gerentes como um assistente para controlar e gerenciar o reuso de software na organização. Um repositório bem projetado pode, por exemplo, ajudar as organizações a mapearem a utilização dos seus componentes com o objetivo de alinhar iniciativas, projetos e auxiliar a medição do impacto provocado pelo reuso nos negócios. Alguns exemplos de informações que podem ser tratadas incluem: quais projetos estão acessando mais o repositório, quais os que precisam de mais incentivo, que tipos de ativos estão sendo mais procurados, que ativos foram procurados e não encontrados, etc.

Motivado por este problema, Banker et al. [Banker et al., 1993] propôs o uso de ambientes CASE baseados em repositório como uma solução para tornar a coleta de métricas uma ferramenta a ser utilizada na análise de reuso de software no nível de repositório - o que eles denominaram de "avaliação de repositório".

Pretendendo provar a eficiência da “avaliação de repositório”, eles apresentaram uma pesquisa que identificou, durante dois anos, os níveis de reuso de duas grandes organizações que usavam uma ferramenta CASE baseada em repositório para manter o software produzido por elas e toda informação relevante a respeito do mesmo.

Os resultados obtidos, indicaram que o uso de repositórios de software integrados com ambientes CASE podem ajudar significantemente o gerenciamento de reuso no nível organizacional e, ao mesmo tempo, tornar possível responder questões como:

- Que tipos de objetos são mais reusados?
- Sob que condições o reuso acontece mais facilmente?

Portanto, a partir desses resultados pode-se concluir que repositórios de reuso devem também oferecer serviços que permitam aos gerentes monitorar o

reuso de software através de múltiplos projetos e, para isso, faz-se necessário que um conjunto seja disponibilizado pelo repositório.

A análise de métricas pode indicar diversos aspectos úteis sobre reuso e podem auxiliar gerentes e desenvolvedores a reduzir custos. Em [Kernebeck, 1997], Kernebeck oferece uma introdução bastante útil a respeito de gerenciamento de repositório de componentes e análise de reusabilidade de componentes de software através de métricas. Kernebeck mostra que a análise de reusabilidade permite-nos determinar o esforço necessário de se reutilizar um componente, o que nos ajuda a decidir entre reusar um dado componente e começar uma nova implementação sem o uso do mesmo.

Em resumo, é possível monitorar as atividades de reuso através de repositórios de componentes com o intuito de reduzir os custos associados a um processo sistemático de reuso de software.

2.1.3 Repositório como um promotor de reuso

O sucesso de um programa de reuso depende do nível de conhecimento que uma organização tem a respeito do processo de reuso adotado. Conseqüentemente, é extremamente importante disseminar a cultura de reuso por toda organização [John & Spiros, 1996].

Sendo assim, repositórios podem prover visibilidade aos seus artefatos através da demonstração de valor de um artefato para desenvolvedores e gerentes. Além disso, eles podem ser usados para disseminar notícias sobre iniciativas de reuso, melhores desenvolvedores de ativos reutilizáveis, componentes mais reusados, entre outros.

2.1.4 Repositório como um assegurador de qualidade

Um problema que surge quando se reutiliza software é como ter certeza da qualidade dos artefatos reusados [Kernebeck, 1997]. Neste contexto, repositórios podem facilitar o objetivo corporativo de manter a qualidade do software

desenvolvido através da disponibilização de artefatos aprovados por um processo de certificação suportado pelo repositório.

A certificação de software, além de ser possível e diretamente aplicável à indústria, proporciona benefícios diretos que podem ser observados em termos de retorno de investimento, qualidade, manutenibilidade, entre outros fatores. Adicionalmente, a certificação é parte fundamental para o sucesso de reutilização de software [Alvaro, 2006].

Na literatura, diversos trabalhos que exploram sistemas de repositório aplicados no contexto de reuso podem ser encontrados. Uma revisão de alguns destes trabalhos é feita na próxima seção.

2.2 Trabalhos na Academia

Origem

Desde 1968, durante a Conferência de Engenharia de Software da NATO, o reuso de software foi anunciado como uma possível maneira de superar a crise de software. Em tal conferência, McIlroy publicou um dos primeiros artigos sobre reuso de software: “Mass Produced Software Components" [McIlroy, 1968]. Neste trabalho, McIlroy discutiu sobre **catálogos padronizados de rotinas**, classificadas por precisão, robustez, desempenho, limites de tamanho e tempo de ligação dos parâmetros. Desde então, de acordo com Lucrédio et al. [Lucrédio et al., 2004], muitas pesquisas, baseadas nas idéias de McIlroy, têm explorado os conceitos de biblioteca e repositório. Mili et al. [Mili et al., 1998] discutem cerca de 50 abordagens para este problema. Dentre estas abordagens, um foco especial é dado aos esquemas de classificação que são usados para armazenar os elementos de um repositório.

Repositórios com classificação baseada em *facetas*

O trabalho de Prieto-Diaz e Freeman [Prieto-Diaz & Freeman, 1987] deu uma contribuição importante para esta área. Neste trabalho, foi proposto o esquema baseado em *facetas* para classificar componentes de software (ex.: linguagem de programação, sistema operacional, domínio de negócio, entre outros). O esquema de facetas foi baseado nas premissas que coleções de componentes reutilizáveis são muito grandes e estão continuamente crescendo, e que há grandes grupos de componentes similares. Nesta abordagem, um número limitado de características (facetas) que um componente pode ter é definido. Segundo Prieto-Diaz e Freeman, facetas podem ser consideradas como perspectivas, pontos de vista, ou ainda dimensões de um domínio particular. Logo, um conjunto de possíveis palavras-chave é associado a cada faceta e, para descrever um componente, uma ou mais palavras-chave são escolhidas para cada faceta.

Repositórios evolutivos

Outros pesquisadores têm explorado diferentes aspectos de suportar reuso com repositórios. Em [Henninger, 1997], por exemplo, Henninger apresenta uma abordagem que utiliza uma estrutura mínima de repositório para apoiar o processo de busca por componentes de software. A abordagem é demonstrada por um par de protótipos: *PEEL*, uma ferramenta para identificar componentes reutilizáveis de maneira semi-automática; e *CodeFinder*, um sistema de recuperação que permite modificar representações de um componente enquanto os usuários estão procurando informação. Porém, os protótipos apresentados estão limitados a um repositório que consiste apenas de funções, variáveis, macros e constantes LISP usadas no Emacs.

A Internet como repositório

Protocolos (http, ftp, pop ...) e tecnologias (servidores web, de e-mail, ...) abaixo do termo Internet oferecem uma infra-estrutura poderosa, barata e aberta que pode se tornar o *backbone* de um repositório de reuso [Ezran el al., 2002]. De acordo com [Seacord et al., 1998], a infra-estrutura da Internet provê os meios necessários para consumidores localizarem os componentes disponíveis e então promover reuso. Neste sentido, Seacord et al. apresentam o sistema *Agora*. O *Agora* combina introspecção com engenhos de busca Web para automaticamente buscar e registrar componentes através da Internet, tornando-os disponíveis para reuso.

Mesmo sendo considerado uma evolução dos repositórios específicos, adotando os princípios de repositórios de referências, o sistema *Agora* apresentou muitas limitações. Por basear-se num mecanismo de busca utilizando introspecção computacional, o sistema apresentou limitações tanto semanticamente, quanto na abrangência, uma vez que são recuperados apenas componentes binários.

Repositórios de busca ativa

Recentemente, um novo tipo de repositório tem sido tratado, os *Sistemas de repositórios de reuso com mecanismos de busca ativa*. Ao contrário dos mecanismos convencionais de acesso a informação, onde os usuários explicitamente iniciam o processo de busca, os mecanismos de busca ativa apresentam informações relevantes para os usuários sem a necessidade destes efetuarem requisições ao sistema [Ye, 2001].

No sistema *CodeBroker*, desenvolvido por Ye [Ye, 2001], o sistema infere a necessidade de componentes dos desenvolvedores de software através do monitoramento das interações destes com o ambiente de desenvolvimento. Deste modo, enquanto o desenvolvedor implementa uma aplicação, o sistema apresenta possíveis componentes que executam as tarefas desejadas pelo desenvolvedor. Esta abordagem de sistema de repositório ativo, embora interessante, apresenta alguns problemas. No modelo atual, as inferências são feitas através da assinatura de métodos e comentários JavaDoc. O maior problema é que os componentes estão

limitados a código fonte e não representam modelos de alto nível, logo o nível de reuso é restrito.

Este tipo de repositório parece bastante promissor, porém, uma série de refinamentos ainda deve ser realizado, a fim de oferecer um melhor rastreamento do conceito de componentes, não apenas como código fonte.

Repositórios com suporte a *"rankeamento"*

Outro aspecto recente de pesquisa está relacionado ao "*rankeamento*" de componentes em um repositório de reuso. Repositórios de reuso usualmente aplicam métodos de representação [Frakes & Gandel, 1990] para facilitar os usuários no processo de busca e utilização de componentes. Neste sentido, Pan et al. em [Pan et al., 2004] propuseram uma abordagem para melhorar a interoperabilidade semântica de repositórios de reuso distribuídos, chamada de I*mproved Relevancy Matching and Ranking* (IRMR), a qual se baseia na análise de correlação dos termos usados nos métodos de representação adotados por repositórios. Baseados no método IRMR, os usuários podem recuperar componentes provenientes de múltiplos repositórios via uma visão de representação simples.

Outras abordagens para auxiliar usuários na priorização (*ranking*) de componentes recuperados podem ser encontradas na literatura. Isakowit, por exemplo, propõe um método que faz uso de hypertexto para organizar a recuperação de componentes [Isakowitz & Kauffman, 1996]. A abordagem *Component Rank* [Inoue et al., 2003] utiliza um método para priorizar componentes, analisando suas relações reais de uso e propagando a significância.

Outras questões relacionadas a repositórios

Além destes, existem outros trabalhos focados no uso de repositórios em conjunto com outros aspectos, tais como, métricas, engenharia reversa e gerência de configuração. Em um estudo conduzido por Banker et al. [Banker et al., 1993], por exemplo, foi observado que ambientes CASE baseados em repositório formam uma solução viável para uma coleta sistemática de métricas de atividades de reuso no

nível do repositório, avaliando a sua eficiência. Adicionalmente, as métricas coletadas podem ser usadas para determinar os objetos que provavelmente serão mais usados e os contextos onde o reuso provavelmente acontecerá mais.

Casanova et al. [Casanova et al., 2003], analisa os conceitos de gerenciamento de configuração e versão e os aplica no contexto de DBC usando bibliotecas de componentes. Neste estudo, Casanova et al. descreve a estrutura geral de bibliotecas de componentes seguindo o modelo de configuração proposto por eles.

2.2.1 Considerações

Analisando os trabalhos apresentados, pode-se concluir que existem diversos estudos na literatura envolvendo repositórios de reuso, porém, o foco destes estudos é, quase sempre, em questões relacionadas à busca e recuperação de componentes, conforme também constatado por Lucredio et al. [Lucredio et al., 2004]. Assim, diferentes aspectos, tais como os papéis que um repositório deve desempenhar em um processo de reutilização não são explorados adequadamente. Conseqüentemente, é difícil encontrar uma solução completa na literatura que apresente com detalhes como um repositório de reuso (que suporte os papéis propostos na Seção 2.1) deve ser especificado, projetado e ainda implementado.

A próxima Seção apresenta uma classificação das soluções práticas que comumente são utilizadas como repositórios de reuso pelas organizações.

2.3 Soluções Práticas

Na prática, a maioria das empresas considera que repositórios são apenas um canal de distribuição de artefatos. Deste modo, repositórios representam, geralmente, simples mecanismos para armazenar, buscar e recuperar artefatos. Como conseqüência, diferentes tipos de ferramentas são usadas como uma opção para armazenar artefatos reusáveis e torná-los disponíveis aos desenvolvedores. Tais ferramentas, variam desde o uso de sistemas de gerenciamento de configuração (SGC), até soluções mais elaboradas como gerenciadores de componentes.

De fato, experiências práticas [Ezran el al., 2002] apontaram que muitas empresas analisadas preferem adaptar ferramentas tradicionais de engenharia de software, a comprar uma solução fechada específica de reuso. Isso, muitas vezes, se deve ao fato dessas soluções serem caras e muitas vezes não atenderem por completo as necessidades da organização.

Nesta Seção, as soluções existentes serão listadas e agrupadas por categorias, pois, enquanto ferramentas específicas aparecem e desaparecem continuamente no mercado, categorias são mais estáveis [Ezran el al., 2002]. O objetivo desta Seção é auxiliar a entender e a classificar os tipos de soluções existentes que são freqüentemente adotadas por empresas. A classificação apresentada formará a base para uma comparação das soluções.

Em geral, as soluções existentes podem ser divididas em 2 tipos principais: *Repositórios de Uso Geral* (*General-usage*) e *Repositórios Específicos de Reuso* (*Reuse-specific*).

2.3.1 Repositórios de Uso Geral

Esta classe inclui ferramentas de propósito geral e sistemas usados no desenvolvimento e gerenciamento de aplicações. Esta categoria está dividida em 4 outras sub-categorias, a saber: *Sistemas de gerenciamento de configuração (SGC)*, *Repositórios de metadados*, *Sistemas Colaborativos* e *Repositórios de ferramentas CASE*.

Sistemas de Gerenciamento de Configuração (SGC)

Como gerenciamento de configuração de software é uma disciplina vital no desenvolvimento de software em larga escala, os sistemas de gerenciamento de configuração são comumente utilizados em organizações de desenvolvimento de software [Hass, 2003]. Desta forma, SGC podem ser vistos como a primeira alternativa para repositórios de reuso, já que eles podem ser usados para o armazenamento de artefatos reusáveis. Exemplos de ferramentas que podem ser usadas para este propósito são: *CVS*[2], *PVCS*[3], *Visual Source Safe*[4], entre outras.

Algumas destas ferramentas possuem características importantes, tais como, controle de acesso, controle de mudanças e gerenciamento de versões. Contudo, elas geralmente possuem APIs limitadas que não permitem, de maneira organizada e eficiente, a definição, a publicação e a busca de ativos reusáveis.

Segundo Ezran et al. [Ezran et al., 2002], outro problema com este tipo de ferramenta é que elas gerenciam elementos de menor granularidade (*fine-grained*) em uma base de projeto, enquanto ativos reusáveis podem possuir uma maior granularidade, contendo composição de elementos de múltiplos projetos.

[2] http://www.nongnu.org/cvs/
[3] http://www.serena.com/Products/professional/vm/
[4] http://msdn.microsoft.com/vstudio/products/vssafe/

Repositórios de Metadados

A manipulação com grandes volumes de dados, dentro de algumas empresas, tem se tornado uma realidade extremamente comum. Acoplada a ela surgiu uma crescente demanda de uma forma que garantisse o efetivo uso da informação de maneira produtiva e inequívoca. É neste contexto que se encaixa os repositórios de metadados.

Os repositórios de metadados representam sistemas que geralmente utilizam a tecnologia de *Data Warehouse* para possibilitar a visualização integrada de todo o ambiente de dados de uma empresa, explicitando através de modelos os inter-relacionamentos existentes entre os dados e as associações destes com os processos funcionais, bases de dados, fluxos de informações, infra-estruturas de processamento/comunicação, etc.

A maioria das ferramentas de mercado, que se encaixam nesta categoria, permite criar um ambiente tecnológico para a gestão da informação nas empresas e, consequentemente, facilitar o reuso destas informações. Algumas dessas ferramentas permitem que os metadados sejam capturados de forma automatizada a partir de bases de dados legadas existentes nas empresas. Uma vez armazenados, os metadados podem ser utilizados para, também de modo automático, disponibilizar diretórios de informação, emitir relatórios e alimentar ferramentas *Computer Aided Sotfware Engineering* (CASE). Um bom exemplo de uma ferramenta de mercado deste tipo é o *Rochade* da *ASG*[5].

Apesar de parecer uma solução bastante promissora por permitir o reuso e integração automática de informações, este tipo de solução possui um custo muito elevado. De acordo com uma pesquisa recente realizada pelo *Gartner*[6] [Blechar, 2005], a maioria das empresas interessadas em reuso preferem usar soluções mais específicas e focadas em tipos de dados que representem artefatos de software reutilizáveis.

[5] http://www.asg.com/
[6] http://www.gartner.com/

Sistemas Colaborativos

Estes sistemas incluem *repositórios de compartilhamento de projetos* e *sistemas wiki.*

- *Repositórios de compartilhamento de projetos*: Esta classe de ferramentas representa repositório para compartilhar projetos de software e permite coordenar o desenvolvimento colaborativo de times distribuídos. Em geral, eles possuem um engenho de busca simples, baseado em palavra-chave, para pesquisar projetos e são bem conhecidas nas comunidades de código aberto que compartilham seus projetos na internet. Exemplos de repositório de compartilhamento de projetos incluem: *BerliOS* (BerliOS, 2005), *Java.Net* (Java.net, 2005) e *SourceForge* (SourceForge, 2005). Alguns desses repositórios, *SourceForge*, por exemplo, possui 2 tipos de produto: (1) *web site público* para o desenvolvimento *open source* com máxima transparência e (2) uma *versão corporativa* projetada para operar dentro de um contexto organizacional. O principal problema destes repositórios é o suporte limitado para explorar e pesquisar componentes de software de alto nível, código fonte e algoritmos reutilizáveis;

- *Sistemas Wiki*: Esta categoria representa sistemas que permitem aos usuários adicionar conteúdo, como em um fórum da internet, e também habilitar outros usuários para editar o seu conteúdo. Geralmente, são sistemas web que executam em um ou mais servidores e são úteis para troca de informações através de um esforço colaborativo. Desta forma, eles podem ser usados para aumentar o reuso de conhecimento existente incluindo padrões e boas práticas que podem ser considerados ativos reusáveis. Porém, assim como os repositórios de compartilhamento de projetos, estes sistemas não provêem o suporte adequado para explorar componentes de alto nível, código fonte e algoritmos reutilizáveis.

Repositórios de ferramentas CASE

Esta alternativa consiste em construir repositórios para integrar ferramentas CASE. Estes repositórios podem ser usados para auxiliar diferentes atividades no processo de desenvolvimento de software. Banker et al. [Banker et al., 1993], Banker et al. apresentam a idéia de ambientes CASE baseados em repositório e mostra como eles podem ser usados para suportar o gerenciamento de reuso no nível organizacional. Por outro lado, Blaha et al. [Blaha et al., 1998] analisam o uso deste tipo de repositório como uma infra-estrutura que facilita as tarefas de engenharia reversa.

Em linhas gerais, repositórios de ferramentas CASE devem armazenar arquivos que serão trocados entre ferramentas. Blaha et al. [Blaha et al., 1998], por exemplo, considera modelos de projeto, modelos de análise, mapeamentos e dados de relatórios de aplicação, como exemplos de arquivos reusáveis que podem ser armazenados neste repositório.

Dados tais benefícios, pode-se considerar este tipo de ferramenta como uma boa opção para facilitar o reuso. Contudo, concordando com Ezran et al., a implementação de uma solução deste tipo requer que as ferramentas CASE envolvidas tenham APIs abertas e que também seja necessário definir protocolos comuns e formatos para troca de dados.

De acordo com Ezran et al. [Ezran et al., 2002], alguns exemplos deste tipo de ferramentas incluem: repositórios para integrar ferramentas CASE em uma plataforma específica (*Allegris* da Intersolv, *Enabler* da Softlab, *MS-Repository* da Microsoft) e repositórios dedicados a ferramentas *Upper CASE* (*IBM Rational Rose, Select/OMT, Paradigm Plus*).

2.3.2 Repositórios específicos de reuso

A Segunda categoria das soluções práticas é formada por ferramentas específicas de reuso, que correspondem, como o nome indica, as ferramentas construídas com o objetivo principal de promover a reutilização de software. Nesta categoria, tem-se as *ferramentas de busca em código* e os *gerenciadores de componentes*.

Ferramentas de Busca em Código

Devido às limitações dos sistemas de gerenciamento de configuração, no que se refere ao gerenciamento de ativos reusáveis, algumas empresas optam por implementar funcionalidades adicionais sobre essas ferramentas. De fato, como código fonte é um dos produtos de software mais reusado, muitas soluções intencionadas em buscar código em repositórios existentes tem emergido, cada vez mais, tanto na academia, quanto na indústria.

O mecanismo geral dessas ferramentas é baseado na indexação de código fonte disponível em repositórios de controle de versão, beneficiados pelo fato de que tais repositórios são intensivamente usados no desenvolvimento de projetos de software, e provêem um serviço de busca de código. Algumas dessas ferramentas tais como *Codase*[7] e *Koders*[8], por exemplo, estão disponíveis como serviços na internet e possuem uma base indexada de código fonte proveniente de múltiplos projetos *open source*.

Em geral, estas ferramentas usam um esquema de busca em texto livre e são baseadas em mecanismos flexíveis e adaptáveis onde diversos tipos de repositórios de código fonte tais como repositórios de controle de versão (*CVS, Subversion*) e repositórios de compartilhamento de projetos na internet (*SourceForge, Java.Net*) podem ser "plugados" na ferramenta.

Exemplos de ferramentas nesta categoria incluem: (i) ferramentas da indústria: *Koders e Codase*; (ii) ferramentas da academia: *CodeFinder* [Henninger, 1997], *Agora* [Seacord et al., 1998], *CodeBroker* [Ye et al, 2000], *SPARS-J* [Yokomori et al., 2003], *MARACATU* [Garcia et al., 2005].

[7] http://www.codase.com/
[8] http://www.koders.com/

A principal desvantagem dessas ferramentas é o fato do reuso ser limitado a código fonte, além disso, não possuem mecanismos que garantam a qualidade dos artefatos que manipulam;

Gerenciadores de Componentes

Tipicamente, ferramentas nesta categoria fazem parte de uma solução corporativa para gerenciar um catálogo de componentes de software reusáveis e provêem serviços para publicar, navegar, buscar e recuperar tais componentes. Elas são, certamente, um tipo de banco de dados que prove uma maneira de identificar e categorizar componentes de software para que os mesmos possam ser compartilhados entre vários usuários dentro de uma organização. Algumas delas suportam características mais sofisticadas, tais como, rastreamento de dependência entre os componentes e geração de relatórios que auxiliam no monitoramento do reuso no nível organizacional. Outras possuem também um conjunto de metadados extensível que pode ser adaptado mais facilmente de acordo com as necessidades das organizações.

A principal desvantagem dessas ferramentas é o alto custo associado à sua aquisição. Além disso, as ferramentas de mercado existentes falham em alguns pontos, como, por exemplo, a ausência de suporte a processos de certificação que garantam a qualidade dos componentes disponibilizados. Exemplos deste tipo de ferramentas são: *LogIdex* (LogIdex, 2005) da *LogicLibrary*, *MS-Visual Component Manager* (MS-VCM, 2005) da *Microsoft*, e *Select Component Manager* (SCM, 2005) da *Select Software*.

A Tabela 2.1 apresenta uma comparação inicial entre as soluções de gerenciadores de componentes citadas. Na próxima Seção os critérios dessa análise serão detalhados e novos critérios serão apresentados. Além disso, a análise será estendida para todos os tipos de soluções existentes, a fim de fornecer uma visão geral das alternativas, sem se prender a ferramentas específicas.

Tabela 2.1 - Comparação de gerenciadores de componentes comerciais

Requisitos	Ferramentas		
	SCM	MS-VCM	LoIdex
Browsing	■	■	□
Busca	■	■	□
Relatórios	■	□	■
Publicação de componentes	■	■	□
Processo de certificação			
Suporte a múltiplos repositórios	■		□
Controle de acesso	■	□	□
Serviços de Notícias			

Legenda

- □ Não suporta/não se aplica
- □ Suporta parcialmente
- ■ Suporta satisfatoriamente

Abreviações

SCM – Select component manager **LL** - LogIdex

MS-VCM - MS-Visual Component Manager

A Figura 2.2 resume a taxonomia das soluções que podem ser usadas, na prática, como repositórios de reuso. A categorização apresentada é bastante útil para entender, de uma maneira geral, a variedade das ferramentas existentes.

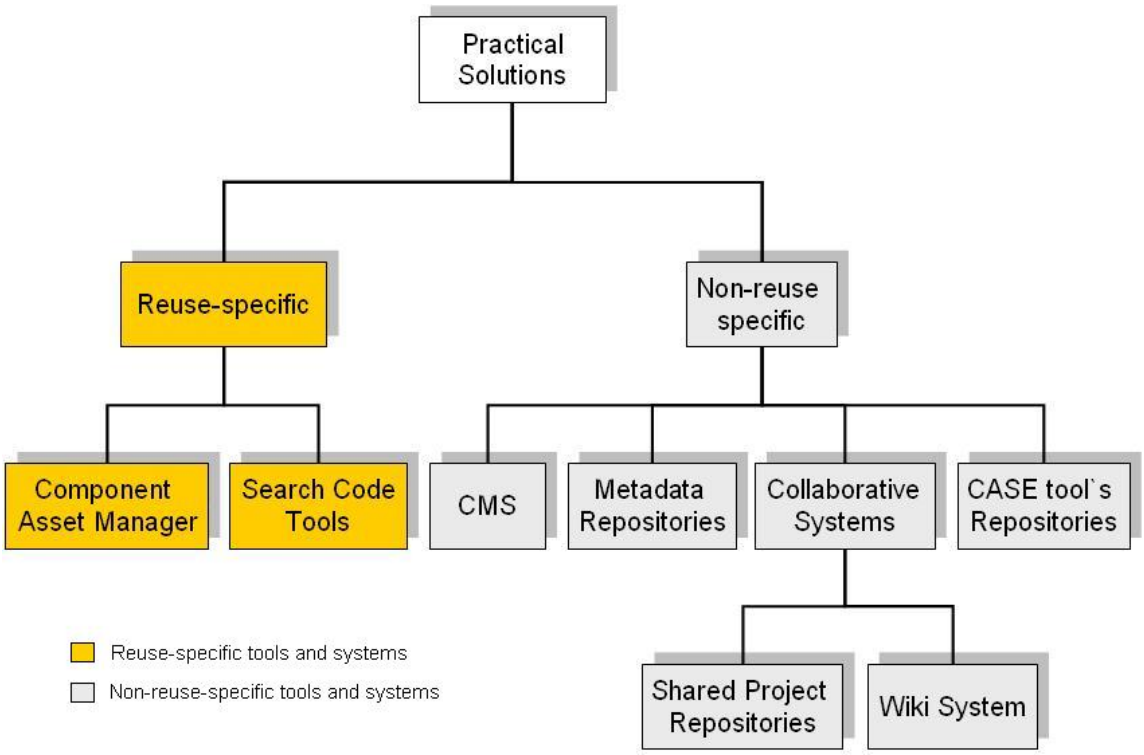


Figura 2.2 - Taxonomia das Soluções

A Seção a seguir apresenta algumas questões que devem ser levadas em considerações ao escolher uma solução de repositório de reuso e reforça o foco da solução proposta.

2.3.3 Considerações das soluções existentes

Após a revisão das soluções, a primeira questão a ser feita é provavelmente: *que tipo de repositório de reuso é o mais adequado para uma empresa?* Na verdade, a resposta para essa pergunta só pode ser feita em conjunto com uma avaliação do contexto em que o repositório será implantado.

Em geral, os fatores a serem considerados ao escolher um repositório de reuso são [Ezran el al., 2002]:

- A quantidade de desenvolvedores, de equipes de desenvolvimento, de unidades organizacionais existentes na empresa;
- A quantidade de domínios diferentes em que esses ativos se encontram distribuídos; e
- A existência na empresa de outros repositórios, tais como, sistemas de gerencia de configuração (SGC) e ferramentas CASE.

Neste sentido, pode-se optar por uma estratégia gradual de implantação de um repositório que, em linhas gerais, funciona da seguinte maneira:

- Se um repositório já existe na organização (SGC ou outros) e ainda não existe nenhum processo sistemático de reuso implantado, pode-se optar inicialmente por desenvolver uma interface para estes repositórios existentes, que ofereça funções específicas para busca e recuperação de artefatos. As ferramentas de busca em código fonte são exemplos deste tipo de solução;
- Por outro lado, quando um processo de reuso já está em andamento e o número de componentes produzidos começarem a crescer, tanto em quantidade, quanto em granularidade, o caminho natural é a implantação de um gerenciador de componentes.

No caso deste trabalho, o foco é voltado para construção de um repositório de reuso para fábricas de software. Neste contexto, geralmente existem vários domínios de aplicações que são desenvolvidas por múltiplas equipes de projetos gerando continuamente componentes reutilizáveis. Na prática, ao invés do termo "componente" foi preferido adotar o termo "ativo", como será visto no próximo Capítulo.

Funcionalidades de um repositório de Reuso

Com o objetivo de efetuar uma comparação das funcionalidades suportadas pelas soluções descritas, foram levantados alguns requisitos que serão utilizados como base para especificar as funcionalidades do repositório a ser proposto.

Para determinar os requisitos foram analisados os papéis de um repositório (descritos na Seção 2.1), algumas considerações práticas, relatadas por desenvolvedores de software, juntamente com as funcionalidades observadas nas ferramentas estudadas. Além disso, foram levados em consideração alguns trabalhos na literatura [Apperly, 2001] [Ezran el al., 2002], que tratam a respeito de requisitos de repositórios aplicados no contexto de reutilização. Uma breve descrição de cada um dos requisitos é feita na Tabela 2.2.

É importante ressaltar que os requisitos listados na Tabela 2.2 serão detalhados no próximo Capítulo.

Tabela 2.2 – Requisitos para comparar as soluções de repositório reuso

#	Requisito	Descrição
1	**Descrição**	É importante estabelecer e manter uma descrição comum para os elementos armazenados no repositório, de modo que todos os membros da organização se tornem familiar com esta descrição. Alguns atributos são fundamentais para classificar e recuperar os elementos, como, por exemplo, as funcionalidades oferecidas por eles.
2	**Inserção**	Produtores de artefatos de software reutilizável precisam torná-los disponíveis para consumo, logo, o repositório deve oferecer serviços que permitam a inserção desses artefatos.
3	**Navegação**	Deve ser possível agrupar os artefatos em diferentes categorias de modo a permitir que os usuários possam navegar pelas categorias existentes.
4	**Busca**	Além da navegação, repositórios com grande quantidade de artefatos devem prover mecanismos de busca que permitam aos usuários encontrar mais facilmente os artefatos que precisam.
5	**Geração de Relatórios**	Os usuários devem ser capazes de gerar relatórios que permitam, entre outras coisas, uma visão geral de como o repositório está sendo utilizado. Exemplos de relatórios: relatórios que indiquem o grau/perfil de colaboração dos usuários e uso dos artefatos, as buscas mais realizadas, os artefatos mais baixados, entre outros.
6	**Notificação de Usuário**	Os usuários devem ser capazes de registrar interesse em diferentes eventos, com o objetivo de no futuro ser notificado quando, por exemplo, um novo artefato de seu interesse for adicionado.
8	**Serviços de Manutenção**	O repositório deve implementar serviços que permitam a manutenção do cadastro de usuários e de outras entidades que são utilizadas pelo sistema.
9	**Gerenciamento de Versão**	Deve ser possível definir e gerenciar múltiplas versões de um mesmo artefato.
10	**Gerenciamento de Dependências**	Deve ser possível que os usuários informem dependências entre os ativos. Estas dependências representam relacionamentos, tais como “usa” ou “é composto por”.
11	**Serviços de Feedback**	Usuários devem ser capazes de fornecer *feedback* a respeito dos artefatos que utilizou. Os *feedbacks* fornecidos permitem a identificação, por exemplo, dos artefatos mais bem avaliados.
12	**Serviços de notícias**	Visando promover a cultura de reuso por toda organização, o repositório deve oferecer serviços que permitam a manutenção e o destaque de notícias relacionadas a reuso, tais como, iniciativas, melhores produtores de artefatos, tipos de artefatos mais reusados, entre outros.
13	**Múltiplas fontes**	O repositório deve suportar o armazenamento de artefatos provenientes de multiplas fontes.
14	**Processo de Certificação**	O repositório deve suportar um processo de certificação que garanta a qualidade dos artefatos disponibilizados.
15	**Métricas**	Os usuários devem ser capazes de definir e capturar métricas a respeito dos artefatos disponibilizados. Exemplo de métricas: esforço e custo de desenvolvimento, LOC (para o caso de código fonte).
16	**Controle de acesso**	O repositório deve possuir mecanismos para limitar o acesso dos usuários a alguns serviços e recursos. E também possibilitar a apresentação (GUI) de diferentes visões do sistema aos diferentes usuários.
17	**Controle de Mudanças**	Os usuários devem ser capazes de requisitar e aceitar mudanças nos seus artefatos. Tais mudanças incluem a reportagem de erros nos artefatos reutilizados.

A Tabela 2.3 classifica as soluções de acordo com os requisitos apresentados. O significado de um campo marcado com "●" é que ferramentas em uma dada categoria ***tipicamente*** suportam o requisito. Por outro lado, um campo em branco significa que a maioria das ferramentas na categoria analisada não satisfaz o requisito especificado de maneira satisfatória.

Tabela 2.3 – Análise das soluções de repositório

Requisitos	Específicas de Reuso		Uso geral				
					Sistemas Colaborativos		
	Gerenciadores de componentes	Busca em código fonte	SGC	Repositórios de Metadados	Repositórios de Projetos Compartilhados	Sistemas Wiki	Repositórios de Ferramentas CASE
Descrição	●						●
Inserção	●		●	●	●	●	●
Navegação	●			●	●	●	●
Busca	●	●		●	●	●	●
Geração de Relatórios	●			●			
Notificação de Usuários						●	
Serviços de Manutenção	●			●	●		
Gerenciamento de Versão			●		●	●	
Gerenciamento de Dependências	●		●				
Serviços de Feedback	●						
Serviços de Notícias						●	
Múltiplas Fontes	●	●		●	●		●
Processo de Certificação							
Métricas	●						
Controle de Acesso	●		●	●	●	●	●
Controle de Mudanças			●		●		

Legenda: ● - tipicamente suportado; em branco: não suportado

A tabela apresentada não pode ser considerada sempre verdadeira, visto que a mesma não é completa e que eventualmente ferramentas específicas de uma categoria podem satisfazer de maneira diferente os mesmos requisitos. Porém, essa análise serve como heurística para ajudar na escolha do tipo de solução de repositório de reuso a ser utilizada em uma empresa.

2.4 Resumo do capítulo

Este Capítulo propôs inicialmente os papéis que devem ser desempenhados por um repositório de reuso. Em seguida, foi feita uma analise das soluções existentes na academia e na indústria. Essa análise teve como objetivo definir o foco e o tipo de solução do repositório proposto neste trabalho. O próximo Capítulo apresenta a especificação dos requisitos, o projeto de arquitetura e a experiência de implementação do repositório de reuso proposto.

3 Repositório de Reuso: Especificação, Projeto e Implementação

Baseado nos papéis de um repositório de reuso, apresentados no Capítulo anterior, um conjunto de requisitos para o repositório proposto foi definido, conforme será apresentado neste Capítulo. O foco da solução proposta consiste em um repositório de reuso que possa auxiliar desenvolvedores na produção e consumo de ativos de software, bem como prover meios que apóiem gerentes na monitoração das atividades de reuso e permita que certificadores de componentes possam garantir a qualidade dos ativos disponibilizados.

Após a definição dos requisitos, a arquitetura do repositório proposto será apresentada e seus módulos serão detalhados. Por fim, a experiência de implementação de um produto real baseado nesta arquitetura será então discutida.

O restante do Capítulo está organizado da seguinte forma: a **Seção 3.1** descreve os requisitos a serem considerados na solução. A **Seção 3.2** apresenta a arquitetura proposta para o repositório seguida do detalhamento interno de seus módulos. A **Seção 3.3** apresenta a experiência de implementação, discutindo o processo utilizado, as tecnologias e *frameworks* adotados e os resultados obtidos. E, finalmente, a **Seção 3.4** resume o repositório proposto.

3.1 Requisitos

A partir da análise dos importantes papéis que um repositório deve desempenhar em um processo de reuso, descritos no Capítulo 2, e das soluções comumente utilizadas na indústria e na academia, pode-se concluir que a maioria das alternativas práticas não suporta eficientemente os papéis desejáveis para um

repositório de reuso. Isto pode explicar porque algumas experiências envolvendo repositórios de reuso têm falhado [Frakes, 1995].

Sendo assim, os requisitos do repositório proposto estão baseados na experiência do autor em reuso e desenvolvimento de software, nas discussões dos membros do grupo RiSE, nas soluções práticas existentes e em alguns trabalhos da literatura que apresentam requisitos para ambientes de reuso [Mascena, 2006], engenhos de busca [Lucrédio et al., 2004] [Garcia et al., 2006] e repositórios [Blaha et al., 1998] [Apperly, 2001] [Ezran el al., 2002]

É importante ressaltar que os requisitos a seguir não representam um conjunto completo e fechado de funcionalidades que devem sempre estar presentes em qualquer repositório de reuso, pois isso dependerá das reais necessidades e do contexto de cada empresa onde o mesmo será utilizado. Contudo, acredita-se que os requisitos identificados podem servir como base para a construção de repositórios que suportem um processo de reuso em organizações com níveis de atividade de reuso mais sistemáticos. Ou seja, organizações onde existe o incentivo para a produção de componentes e a incorporação das idéias de reuso no desenvolvimento [Lynex & Layzell,1998]. A Seção a seguir discute o conteúdo que deve ser armazenado em um repositório. Em seguida, a listagem de funcionalidades, que devem ser satisfeitas pela solução proposta é apresentada e finalmente algumas considerações são feitas a respeito dos requisitos não funcionais que devem ser considerados.

3.1.1 O armazenamento de ativos em repositórios

O desenvolvimento de software envolve um conjunto grande de atividades e diferentes tipos de artefatos são produzidos ao longo deste percurso. Sendo assim, uma empresa que inicia a construção de um repositório de reuso, provavelmente, possui uma quantidade significativa de artefatos de software, tais como: especificação de casos de uso, unidades de programas, código fonte, modelos de projeto e outros documentos que não necessariamente representam um componente de software, mas podem ser reutilizados em outros contextos ou sistemas.

Por esta razão, os elementos suportados em um repositório não podem estar limitados a apenas um tipo de artefato e devem englobar qualquer produto de software que apresente um potencial de reuso significativo. Neste sentido, alguns pesquisadores têm buscado uma definição para *artefatos reusáveis*.

D'Souza and Wills [D'Souza & Wills, 1999], por exemplo, definem *artefatos reusáveis* como sendo qualquer parte do trabalho que possa ser utilizada em mais de um projeto. Eles consideram os seguintes artefatos como candidatos à reutilização:

- Código compilado; objetos executáveis;
- Código Fonte; classes e métodos;
- Casos de testes;
- Modelos e Projetos: frameworks, padrões;
- Interface de usuário; e
- Planos, estratégias e regras arquiteturais.

Portanto, o primeiro passo para determinar a estrutura dos elementos armazenados em um repositório consiste em definir um *modelo genérico* que possa ser capaz de representar diferentes tipos de unidades de software reutilizáveis.

Nesta direção, muitos pesquisadores, quando se referem a uma unidade de software reutilizável, preferem introduzir o conceito de "ativo" (*asset*). Em um sentido geral, um ativo representa algo de valor que uma pessoa ou uma empresa possui, ou seja, um bem. E este conceito pode ser adaptado para o contexto de software, onde então, o termo "ativo" pode ser definido como uma unidade de software reutilizável que captura algum conhecimento de negócio e que possui um valor significativo para uma empresa. Sendo assim, segundo Ezran et al. [Ezran et al., 2002], o termo "componente de software" pode ser visto como um tipo específico de ativo que pode ser executado.

Ainda de acordo com Ezran et al., a representação de um ativo deve incluir duas partes: o conteúdo (*asset body*) e os meta-dados (*asset description*):

- **Conteúdo**: pode ser definido como sendo o conjunto de artefatos relacionados (partes reutilizáveis). Cada artefato é um produto de trabalho do ciclo de vida do software e pode representar a mesma parte do software em diferentes níveis de abstração. Como exemplos de tipos de artefatos podem ser citados: modelo de análise, modelo de projeto, código fonte e código executável;
- **Metadados**: consiste do conjunto de informações a respeito do conteúdo do ativo. Esse conjunto de informações deve incluir atributos, como, por exemplo, identificador do ativo, autor, datas de criação e última atualização do ativo, descrição, conjunto de palavras-chave ou facetas. Esse conjunto de atributos torna possível identificar, descrever e buscar os ativos do repositório.

O modelo genérico do conceito de ativo pode ser representado com o diagrama UML apresentado na **Figura 3.1.**

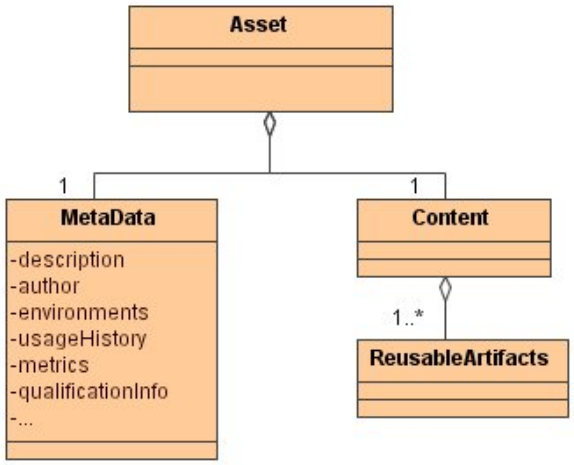


Figura 3.1 – Modelo de ativo simplificado

3.1.2 Funcionalidades do Repositório

Em linhas gerais, um repositório deve ser capaz de identificar ativos, associar uma descrição para cada um deles, manter *links* para os artefatos que os compõem, além de oferecer serviços que suportem as atividades de produtores, consumidores, gerentes e certificadores de componentes. Em detalhes, um repositório deve oferecer as seguintes funcionalidades:

1. **Inserção.**Um usuário (tipicamente um produtor de ativo) precisa tornar seus ativos disponíveis para consumo, logo, um repositório deve oferecer um serviço básico de inserção de ativos. A funcionalidade de inserção deve englobar o armazenamento dos metadados do ativo e também o seu conteúdo. Os metadados cadastrados deverão ser utilizados para organizar o catálogo de elementos do repositório. Já o cadastro do conteúdo deverá suportar tanto a inserção dos artefatos localmente (basicamente uma operação de *upload* dos artefatos), como também a inserção de referências para artefatos remotos, localizados em outras fontes externas, como, por exemplo, repositórios de controle de versão ou até mesmo a internet;
2. **Navegação.** Deve ser possível agrupar os ativos em diferentes categorias e deve existir um mecanismo que permita a navegação pelos ativos através destas categorias. Isto tem por finalidade oferecer uma visão mais simplificada do conjunto de catálogos de componentes presentes no repositório. De acordo Ezran et al., o serviço de navegação é suficiente para ajudar os desenvolvedores a encontrar os ativos que precisam, quando a quantidade de ativos no repositório é relativamente pequena (na ordem de dezenas);
3. **Classificação e busca.** Repositórios com uma quantidade muito grande de ativos devem prover mecanismos de busca que permitam aos usuários encontrar o que precisam, uma vez que a navegação em catálogos não se mostra suficiente neste contexto. Existem várias técnicas que podem ser utilizadas com o objetivo de facilitar o processo de classificação e busca de ativos, essas técnicas incluem busca textual, palavras-chave e classificação baseada em facetas;

- **Busca textual**. Nesta técnica, os usuários devem ser capazes de buscar no catálogo por um *string* de texto contida em qualquer parte da descrição do ativo. O resultado de tal busca é um conjunto de uma ou mais descrições de ativos. Uma variação desta funcionalidade pode também levar em consideração a busca no conteúdo dos ativos;

- **Palavras-chave**. Nesta técnica, cada ativo é descrito por uma ou mais palavras-chave, por exemplo, uma implementação em Java de uma lista de objetos pode ser descrita pelas seguintes palavras-chave: 'código', 'estrutura de dados' e 'Java'. Essas palavras-chave são informadas pelos usuários produtores de ativos e o processo de busca consiste do usuário informar uma ou mais palavras-chave como entrada, e receber zero ou mais descrições de ativos como saída;

- **Classificação baseada em facetas**. Cada ativo é descrito por um conjunto de pares, atributo-valor, exemplificando, a lista de objetos poderia ser descrita por '*Tipo de* ativo: código fonte'; '*Funcionalidade*: estrutura de dados'; '*Linguagem de programação*: Java'. Classificação por facetas [Prieto-Diaz, 1991] é uma técnica mais rigorosa do que palavras-chave. Na busca, o usuário informa um filtro composto de um ou mais pares de atributo-valor, e recebe como saída, zero ou mais descrições de ativos. Uma variação mais poderosa da classificação por facetas define estrutura de árvores para os valores da faceta [Ezran et al., 2002]. Por exemplo, a faceta '*Linguagem de programação*' poderia ter os valores 'procedural' ou 'orientada a objetos'. O valor procedural poderia ser especializado como 'Pascal', 'Fortran' e 'C' e o valor 'orientada a objetos' como 'Smalltalk', 'C++' e 'Java'. O processo de busca nesse caso deve levar em consideração qualquer nível da árvore. Por exemplo, a busca com '*linguagem de programação* = procedural' retorna todos os ativos que possuem os valores 'Pascal', 'Fortran' e 'C', e a busca com '*linguagem de programação* = Java' restringe o escopo de busca aos ativos que possuem o valor 'Java'.

É importante ressaltar que o requisito de busca é extremamente crucial e complexo em um ambiente de repositório. Um dos principais pesquisadores envolvidos com repositórios, Ali Mili [Mili et al., 1998], resume as pesquisas envolvendo mecanismos de armazenamento e busca de artefatos do seguinte modo: "*Apesar das contínuas pesquisas, o problema do armazenamento e recuperação de artefatos de software continua como um problema em aberto. Mesmo existindo uma grande variedade de soluções para este problema, nenhuma solução oferece a combinação de eficiência, precisão e facilidade de utilização para permitir otimizar a reutilização de software*".

Em um trabalho mais recente, Lucrédio et al. apresentam um exaustivo estudo sobre busca e recuperação de artefatos em sistemas de repositório e ratificam a conclusão de Ali Mili [Lucrédio et al., 2003], [Lucrédio et al., 2004].

4. **Geração de Relatórios**. O repositório deve prover serviços para a geração de relatórios que permitam, entre outras coisas, obter uma visão geral de como o repositório está sendo utilizado. Como exemplos de relatórios que devem ser suportados, pode-se citar: relatórios que indiquem o grau/perfil de colaboração dos usuários e uso dos componentes, as buscas mais realizadas, os componentes mais baixados, os componentes mais recentes, entre outros. Um detalhamento dos possíveis tipos de relatórios com exemplos dos mesmos, podem ser vistos no Apêndice B.
5. **Notificação dos usuários.** Os usuários devem ser capazes de registrar interesse em diferentes eventos ou conjunto de informações do repositório, com o objetivo de, no futuro, serem notificados quando, por exemplo, um novo ativo for adicionado, novas versões de ativos existentes forem criadas e informações em geral do repositório, como, por exemplo, a adição de notícias ou artigos em geral;
6. **Serviços de Administração.** O repositório deve implementar serviços que permitam a manutenção do cadastro de usuários e de outras entidades que

são utilizadas pelo sistema. Estas entidades podem incluir os tipos de artefatos, tipos de ativos, os classificadores (facetas), entre outras.

7. **Gerenciamento de versão.** O repositório deve ser capaz de armazenar múltiplas versões de um mesmo ativo, a fim de permitir que os usuários sejam capazes de recuperar versões antigas de um ativo e manter versões variantes, ou seja, implementações alternativas do mesmo ativo;
8. **Gerenciamento de dependência.** Deve ser possível que os usuários informem dependências entre os ativos. Estas dependências representam relacionamentos, tais como, "usa" ou "é composto por";
9. **Suporte a *Feedback*.** O sistema deve permitir que os usuários dos ativos registrem suas impressões a respeito dos componentes que utilizaram. Os *feedbacks* permitem, por exemplo, a identificação dos componentes mais bem avaliados e também o rastreamento de uso dos ativos (o contexto em que o ativo foi usado, os problemas encontrados, entre outros). O mecanismo de *feedback* também pode ser utilizado para gradualmente melhorar a qualidade do resultado das buscas feitas no sistema. Um exemplo citado por Mascena [Mascena, 2006] é o fato do mecanismo de busca atribuir maior prioridade aos ativos que foram reusados com sucesso por outros programadores. Além disso, o conjunto de *feedbacks* pode ser usado juntamente com políticas de incentivo, como, por exemplo, bonificação para os desenvolvedores dos componentes mais bem avaliados. Uma discussão útil a respeito de políticas de incentivo aplicadas a reutilização pode ser encontrada em [Poulin, 1995];
10. **Serviços de *marketing*.** Visando promover a cultura de reuso por toda organização, o repositório deve oferecer serviços que permitam a manutenção e o destaque de notícias relacionadas a reuso, tais como, iniciativas, melhores produtores de ativos, componentes mais reusados, entre outros;
11. **Suporte a múltiplas fontes de ativos**. O repositório deve suportar múltiplas fontes de ativos, isto é, deve ser possível armazenar ativos cujos artefatos (conteúdo) estejam armazenados em diferentes fontes externas. Isto facilita a integração com diferentes tipos de fontes de ativos que possam

existir no contexto onde o repositório será implantado, como, por exemplo, repositório de controle de versão e diretórios compartilhados na rede;

12. **Suporte a um processo de certificação**. O repositório deve oferecer um serviço de certificação que permita garantir a qualidade dos ativos disponibilizados. O *workflow*, que é parte do processo de certificação definido está descrito no Apêndice C. Duas considerações importantes devem ser ressaltadas quanto à certificação: i. após certificado, qualquer alteração deve gerar uma nova versão do ativo; ii. o repositório exibirá tanto os ativos certificados, quanto os não certificados;

13. **Métricas**. O repositório deve oferecer serviços que permitam aos usuários informar e capturar métricas a respeito dos ativos do repositório. A análise de métricas pode indicar uma série de informações úteis a respeito dos ganhos com reuso (conforme pode ser visto no relatório que mede o retorno do investimento no Apêndice B);

14. **Controle de acesso**. Cada funcionalidade acima deve estar disponível apenas para perfis de usuários selecionados. Por exemplo, a operação de 'Inserção' deverá ser executada apenas por produtores de ativos, a 'Navegação' para todos os usuários; e

15. **Controle de mudanças**. Procedimentos para solicitar, discutir, aceitar e implementar mudanças nos ativos devem ser definidos e reforçados através de suporte automático, ou seja, devem existir serviços no repositório que permitam aos usuários, por exemplo, o cadastro de *bugs* e melhorias nos ativos reutilizados.

Além das funcionalidades apresentadas, existem alguns requisitos não funcionais que devem ser considerados, tais como, usabilidade, escalabilidade, desempenho, distribuição, integração com outros ambientes e ferramentas, entre outros. A próxima Seção discute a respeito dos perfis de usuários que devem ser tratados no repositório.

3.1.3 Perfis de usuários

Normalmente, em uma organização, um grande número de pessoas, de diferentes projetos, possuem acesso a um repositório de reuso em várias fases do processo de desenvolvimento de software, seja para armazenar ou para recuperar informações. Com o objetivo de satisfazerem suas tarefas e atribuições, cada uma destas pessoas possui necessidades que não necessariamente precisam ser atendidas por todos os serviços oferecidos pelo repositório, mas somente por uma parte destes. Desta forma, interfaces especializadas, que atendam e ofereçam apenas as partes e funcionalidades do repositório necessárias para a execução das tarefas correntes, precisam ser oferecidas. É neste contexto que se encaixa a idéia de perfis dos usuários do repositório.

Feldman et al. [Feldman et al., 2000] discutem o conceito de perfis aplicados a sistemas de repositório de reuso, discutindo os interesses e as atribuições de cada perfil. Além disso, os perfis apresentados para usuários de um repositório de reuso são mapeados nos papéis mais comumente encontrados na Engenharia de Software (ES).

Inspirado nas idéias de Feldman et al., foi definido um conjunto de 4 (quatro) perfis básicos de usuários que devem ser suportados pelo repositório proposto. Os perfis considerados foram divididos em duas categorias, a saber: **perfis técnicos** e **perfis de gerenciamento**. Os perfis técnicos são representados pelos consumidores e produtores de ativos, já os perfis de gerenciamento correspondem aos administradores do repositório e aos usuários responsáveis pela certificação dos ativos. A Tabela 3.1 apresenta os 4 (quatro) perfis a serem adotados no repositório (produtor, consumidor, administrador e certificador), seus principais interesses e os papéis da engenharia de software que comumente estão associados a cada um desses perfis.

Tabela 3.1 - Perfis básicos do repositório

	Perfis básicos de usuários do repositório	Principais Interesses	Papéis associados da ES
Perfis Técnicos	Consumidor	Buscar e (re)-usar os ativos do repositório	Programador, Engenheiro de requisitos, testador, analista de negócios, arquiteto de software
	Produtor	Criar e publicar ativos	
Perfis de Gerenciamento	Administrador	Manter o repositório e monitorar as atividades de reuso na organização	Gerente de projetos, gerente de produtos, planejador de projetos, administrador de sistemas
	Certificador	Manter a qualidade dos ativos disponíveis no repositório	SQA (*Software Quality Assurance*)

3.1.4 Resumo dos requisitos

A partir dos requisitos listados na seção anterior, fica evidente que o principal objetivo é prover os usuários com um conjunto completo de funcionalidades focadas em facilitar as atividades de produção, consumo, gerenciamento e certificação dos ativos de software reusáveis em uma organização.

A próxima Seção discute a arquitetura proposta para suportar esses requisitos.

3.2 Arquitetura do repositório

A arquitetura de software tem desempenhado um papel fundamental no desenvolvimento de sistemas de software. Ela oferece um maior entendimento da aplicação por dividi-la em um conjunto de componentes que interagem entre si para realizar parte de uma ou várias funcionalidades do sistema [Garlan & Shaw, 1993].

De acordo com [Clements et al., 2002b], uma das formas mais utilizadas de documentar a estrutura de um sistema é decompô-lo em módulos. A divisão de um

sistema em módulos menores e menos complexos provê uma maneira de fazer com que o desenvolvedor concentre seus esforços em pedaços especializados em algum serviço e garanta que eles sejam completos e validados. Um módulo, neste caso, representa uma unidade de software que implementa um conjunto de funcionalidades. Sendo assim, a visão em módulos de uma arquitetura pode ser utilizada como a base para determinar como os requisitos de um sistema são suportados através das responsabilidades dos módulos que constituem a aplicação.

Sobre esta motivação, esta Seção apresenta a definição geral dos módulos da arquitetura do repositório proposto. O principal objetivo é satisfazer da melhor maneira o conjunto de requisitos definidos na Seção 3.1. Para isso será apresentada inicialmente uma visão unificada de como o sistema foi estruturado e em seguida seus principais módulos serão detalhados.

3.2.1 Visão geral

A Figura 3.2 apresenta uma visão geral da arquitetura do sistema. Como mencionado na Seção 3.1.3, o repositório proposto possui um controle de acesso baseado em perfis, os quais definem quatro papéis de usuários: consumidor, produtor, administrador e certificador. Estes usuários estão representados no topo da Figura e utilizam ferramentas (tipicamente um *browser* ou uma aplicação *desktop*) para acessar os serviços do repositório através da camada de acesso (*access layer*).

Esta camada tem como função permitir a interação das ferramentas dos usuários com os *módulos de negócio* do repositório. É nela onde são definidos os pontos de acesso ao sistema, os quais podem incluir, por exemplo, um serviço `http` (aplicação web) ou uma interface de um serviço web (*web service*) para acesso ao sistema via *SOAP*. Para o caso de uma aplicação web, a camada de acesso pode incorporar os papéis de uma camada de apresentação, pois, neste caso, seria responsável por gerenciar o processamento das requisições e posterior apresentação dos resultados.

Os três principais módulos de negócio (*Business*) do repositório são: (1) Módulo de Produção (*Production),* (2) Módulo de Consumo (*Consumption*) e (3) Módulo de Gerenciamento (*Management*).

O *módulo de produção* é responsável por apoiar as atividades dos produtores de ativos, através dele deve ser possível, por exemplo, publicar novos ativos no repositório e manter os já existentes.

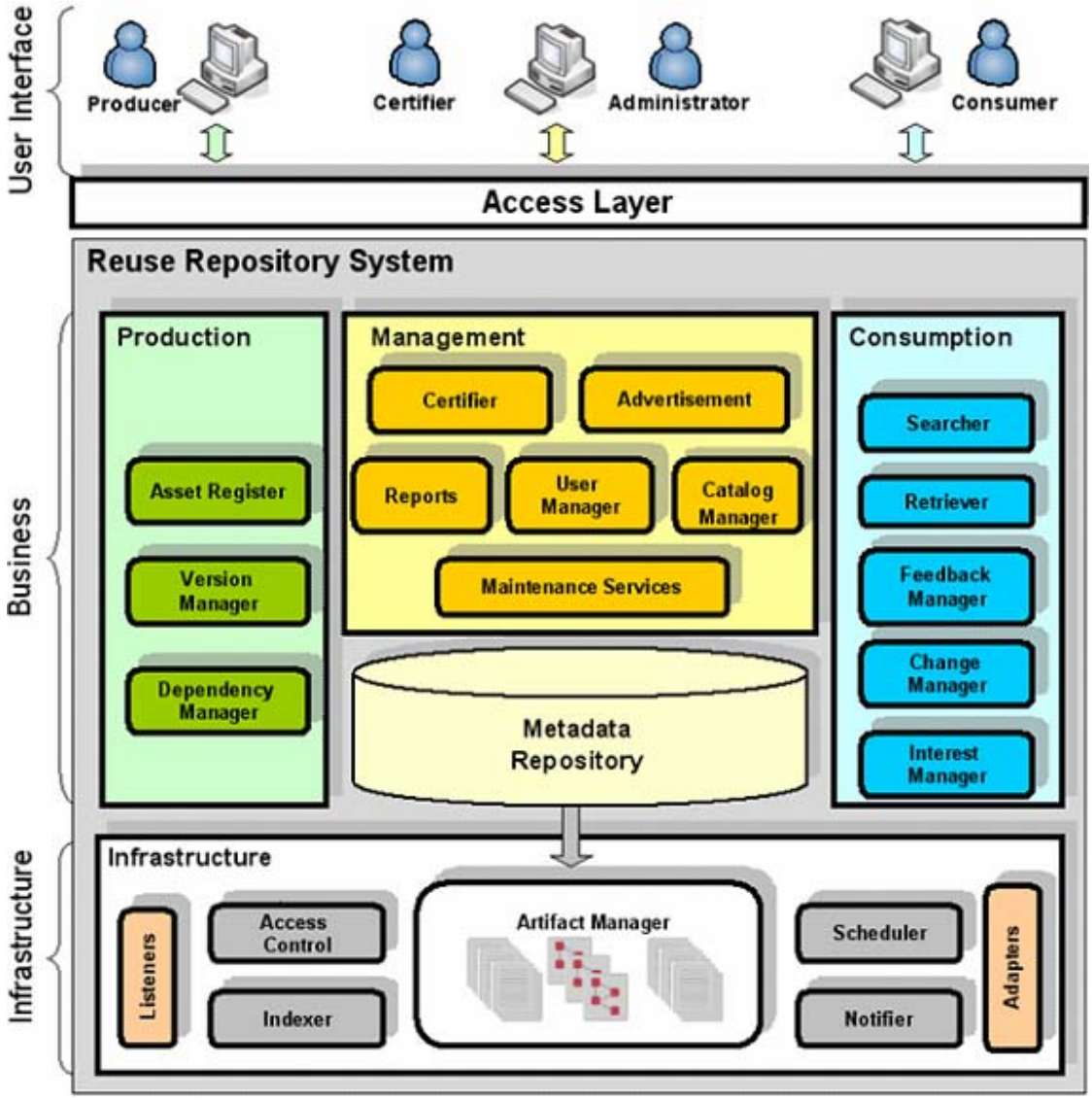


Figura 3.2 – Visão geral da arquitetura

O *módulo de consumo* disponibiliza os serviços necessários para que os usuários interessados em reusar ativos (consumidores) consigam encontrar os artefatos que desejam.

Por sua vez, o *módulo de gerenciamento* age como uma ferramenta de apoio a ser utilizada para monitorar as atividades de reuso na organização e garantir a qualidade dos ativos disponibilizados. Este módulo, portanto, é utilizado por usuários com perfil de administrador e certificador.

O elemento central da arquitetura, que é utilizado pelos três módulos de negócio, é o *Metadata Repository*. Ele funciona como uma base de dados compartilhada, que disponibiliza serviços para armazenar e recuperar os metadados dos ativos disponibilizados, bem como as informações necessárias para o funcionamento e o gerenciamento do repositório como um todo.

Apesar de estar em destaque na Figura 3.2, o *Metadata Repository* faz parte do módulo de infra-estrutura (*Infrastructure*) que, além dele, é formado por um conjunto compartilhado de componentes e serviços que podem ser utilizados por todos os outros módulos do repositório.

Uma vez definida a idéia geral dos módulos do repositório, o passo seguinte consiste em detalhar essa visão, apresentando como cada um dos módulos principais está organizado internamente. Para isso, as próximas Seções descrevem com mais detalhe as responsabilidades de cada módulo e apresentam diagramas que explicitam as dependências dos seus componentes e sub-módulos internos. É importante ressaltar que a abordagem de especificação utilizada foi inspirada no processo de divisão em módulos discutido por Bass et al. em [Bass et al., 1998]. Nesta abordagem, a estrutura interna de cada módulo é apresentada através dos sub-módulos que o compõe e suas relações com outros elementos. Para documentar os diagramas, foram utilizados alguns elementos da linguagem de modelagem UML (*Unified Modeling Language*) [Larman, 2003]. A UML tem emergido como a notação padrão para documentação de arquitetura de software [Clements et al., 2002b]. Outra possibilidade poderia ser a utilização de uma ADL

[Clements, 1996], entretanto, os trabalhos envolvendo ADL ainda não mostraram sua aplicabilidade em cenários industriais, como o caso do presente trabalho.

A próxima Seção inicia o detalhamento do módulo de infra-estrutura, já que ele é utilizado como base pelos outros módulos.

3.2.2 Módulo de Infra-estrutura

O módulo de infra-estrutura forma uma camada de serviços que são compartilhados por todo o repositório. Estes serviços incluem desde a persistência dos dados do sistema, até operações que controlam o acesso aos recursos do repositório. A Figura 3.3 apresenta a estrutura interna deste módulo, que é formada pelos seguintes elementos: *MetadataRepository, ArtifactManager, Indexer, AccessControl, Notifier* e *Commons*. Cada um deles é descrito a seguir.

MetadataRepository. O principal componente do módulo de infra-estrutura é o *MetadataRepository*. Este componente representa uma abstração do mecanismo de persistência a ser utilizado pelos outros módulos do repositório. Sua interface (*IMetadataRepository*) possui serviços genéricos para o armazenamento e recuperação dos dados das entidade do sistema, sendo a principal delas, a entidade que representa um ativo reutilizável.

No *Metadata Repository*, os ativos estão agrupados em múltiplos *catálogos*, que juntos, formam um repositório virtual único. Assim como diretórios, os catálogos podem ser definidos hierarquicamente. Com isso, podem ser usados para representar a estrutura de uma organização (departamentos, equipes de projeto, unidades de negócio, entre outros) ou então domínios de aplicações (domínio financeiro, domínio hospitalar, etc). Isso ficará a cargo do contexto onde o repositório será implantado.

Conforme visto na Seção 3.1.1, um ativo é formado por um conjunto de metadados a respeito dos artefatos de software que compõem o seu conteúdo. Logo, é importante ressaltar que apenas os seus metadados são mantidos no

MetadataRepository, já seus artefatos são armazenados no componente *ArtifactManager*, o qual pode ser visto como uma fonte de artefatos do repositório.

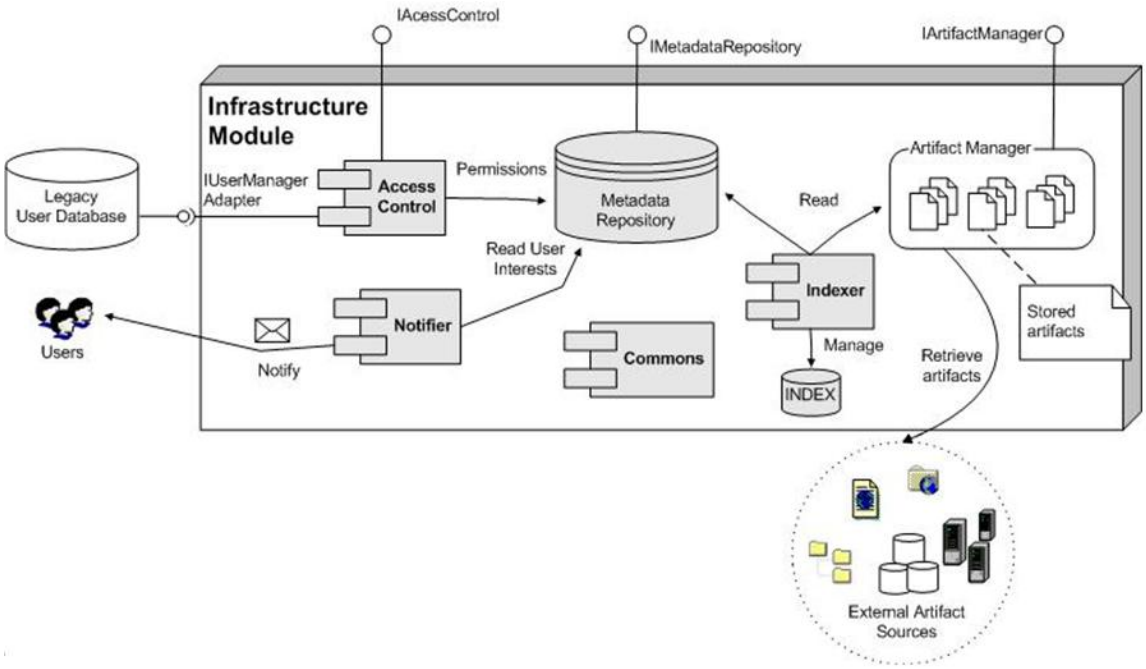


Figura 3.3 – Módulo de Infra-estrutura

ArtifactManager. Para ser efetivo, um repositório de reuso não deve apenas permitir que os artefatos reusáveis sejam sempre armazenados localmente no repositório. Ao invés disso, ele deve ser capaz de referenciar outros repositórios mantidos pelas ferramentas que criaram os artefatos: sistemas de controle de versão, ferramentas de modelagem UML, entre outras [Ezran et al., 2002].

Sendo assim, o *ArtifactManager* pode ser usado tanto como um repositório que armazena localmente os arquivos, quanto como um mecanismo de abstração das diferentes fontes de informação disponíveis. Esta abstração é útil quando o usuário opta por armazenar apenas uma referência para um artefato remoto, ao invés de persistir seu conteúdo no repositório. Exemplificando, um usuário pode cadastrar os metadados de um renomado *framework* disponível na internet e, ao invés de salvar o seu conteúdo no repositório, optar por armazenar as *urls* para os arquivos disponibilizados no site de tal *framework*.

Indexer. Com o intuito de prover mecanismos de busca mais sofisticados, os ativos armazenados no repositório passam por um processo de indexação. A indexação corresponde a primeira etapa de um processo de recuperação de informação (*information retrieval*). Ela é responsável por manipular os dados a serem consultados e criar estruturas especializadas que representem esses dados, com o objetivo de permitir um acesso mais rápido as informações contidas. A estrutura de dados mais comumente utilizada para esse propósito é conhecida como *índice* [Baeza-Yates & Ribeiro-Neto, 1999].

Na arquitetura, o *Indexer* é o sub-módulo responsável pelo processo de indexação, que engloba tanto a indexação dos metadados dos ativos, quanto do seu conteúdo (artefatos armazenados). Durante a fase de indexação, os metadados e o conteúdo dos ativos são percorridos (*parsing*), de acordo com seus tipos e formatos, e analisados antes de serem indexados realmente.

O *parsing* é responsável por interpretar os tipos e os formatos dos dados a serem indexados e mapear o conteúdo desses dados para uma representação normalizada, enquanto a fase de análise é responsável por interpretar esta representação normalizada e determinar o que é relevante para ser indexado.

Segundo [Mascena, 2006], algumas abordagens podem ser executadas durante a fase de análise com o objetivo de maximizar a performance do mecanismo de busca que usará o índice gerado. Exemplos destas abordagens incluem *manipulação de identificadores* [Michail & Notkin, 1999] [Jensen, 2004], *redução de palavras* [Rijsbergen et al., 1980], conversão de palavras para a sua representação fonética [Gospodnetic & Hatcher, 2004], entre outros.

Access Control. É o sub-módulo responsável pela autenticação e autorização dos usuários do sistema. O *AccessControl* pode ter seu processo de autenticação integrado a uma base legada de usuários da organização. Esta integração visa proporcionar um ambiente menos intrusivo e é feita através de um adaptador que pode facilmente ser implementado, evitando, assim, o re-cadastro dos mesmos.

Este sub-módulo também é responsável por verificar o acesso dos usuários aos diferentes catálogos de ativos existentes. Esse controle deverá ser feito através

de grupos de usuários, que determinarão visões diferentes dos catálogos e limitarão a busca de ativos dentro da alçada de cada usuário.

Notifier. Outro serviço importante do módulo de infra-estrutura é o envio de notificações para os usuários do repositório (requisito #6). Segundo [Apperly, 2001], o recebimento de notificações via e-mail é a melhor abordagem a ser utilizada, pois possibilita que arquivos sejam anexados a mensagem.

Este serviço é tratado pelo sub-módulo *Notifier* que monta o conteúdo das notificações a partir dos interesses registrados pelos usuários. Geralmente, as notificações são usadas para informar aos consumidores de ativos quando novos componentes ou informações importantes estão disponíveis no repositório. Elas também podem ser úteis para notificar produtores a respeito de solicitações de mudanças em seus ativos e deixá-los informados sobre o processo de certificação dos seus componentes. É importante que, pelo menos, um link para o ativo no repositório seja enviado.

Commons. Além dos serviços descritos acima, o módulo de infra-estrutura do repositório agrupa uma coleção de componentes comumente encontrados em sistemas desse tipo. Estes componentes estão representados na Figura 3.3 pelo sub-módulo *Commons* que possui funcionalidades como: mecanismos para log de erros e de execução, tratamento de erros, *Scheduler*, entre outros.

Dando continuidade ao detalhamento interno dos módulos, a Seção a seguir apresenta o módulo de Produção, que é um dos módulos de negócios do repositório.

3.2.3 Módulo de Produção

O módulo de Produção (*Production)* é responsável por disponibilizar os serviços a serem utilizados pelos produtores de ativos. Estes serviços devem incluir operações que permitam a manutenção tanto dos metadados do ativo, quanto do seu conteúdo (artefatos). A visão interna do módulo de produção é apresentada na Figura 3.4 e em seguida as responsabilidades de seus principais sub-módulos –

AssetExtractor, *AssetRegister*, *VersionManager*, *DependencyManger* - são descritas.

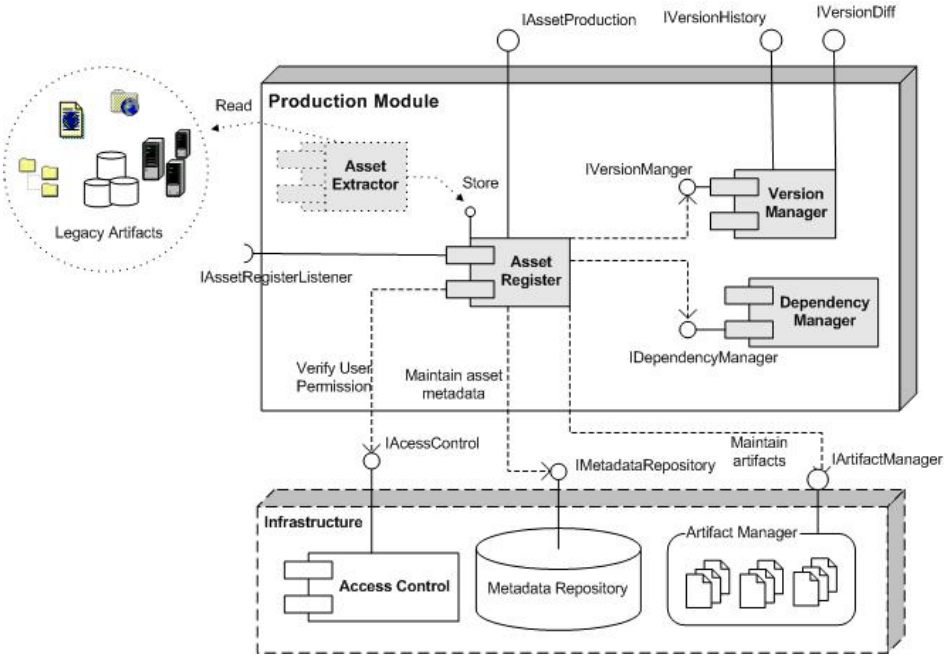


Figura 3.4 – Módulo de Produção

AssetRegister. O *Asset register* é o sub-módulo de produção mais importante, pois é o responsável direto pela interface *IAssetProduction*. Esta interface permite que os produtores executem atividades de manutenção nos seus ativos como: (1) gerenciar *versões* de um ativo, (2) especificar *dependências* (ativos relacionados), (3) manter os *artefatos* disponibilizados, (4) fornecer *métricas* a respeito dos ativos produzidos, e (5) associar descritores e *classificadores* (facetas) que caracterizem seus ativos e conseqüentemente facilitem a busca pelos mesmos.

Para manter os dados envolvidos nas funcionalidades listadas, o *AssetRegister* utiliza os componentes *MetadataRepository*, que mantém os metadados dos ativos, e *ArtifactManager* que é responsável pela manutenção do conteúdo do ativo. Estes componentes fazem parte do módulo de infra-estrutura, explicado na Seção 3.2.2. É importante observar, que os dados manipulados pelo *AssetRegister* correspondem às informações presentes no modelo de classes que especifica a estrutura de um ativo. Os detalhes deste modelo serão vistos posteriormente.

Além das dependências como o *MetadataRepository* e o *ArtifactManger*, o *AssetRegister* utiliza o sub-módulo de controle de acesso provido pela infra-estrutura. Essa dependência existe para que seja possível verificar se o usuário (produtor) possui a permissão adequada para alterar o catálogo de ativos que deseja.

Durante a manutenção dos ativos no repositório, pode ser necessário o acionamento de algumas operações, como, por exemplo, a atualização do índice de busca quando novos ativos são inseridos no repositório. Para isso, o *AssetRegister* disponibiliza a interface *IAssetRegisterListener* que pode ser implementada para capturar os eventos gerados durante a manutenção dos ativos. Diversas implementações para esta interface podem ser configuradas no sistema sem a necessidade de alteração no *AssetRegister*.

A seguir serão descritos os sub-módulos *VersionManager* e *DependencyManager* utilizados internamente pelo *AssetRegister*.

Version Manager. O *VersionManager* é utilizado para auxiliar o processo de gerenciamento das versões de um ativo. Para isso, sua interface *IVersionManager* disponibiliza serviços que permitem a criação, a cópia, a remoção, a listagem e a alteração das versões de um ativo. Além disso, ele realiza as validações de consistência a respeito da manutenção de versões.

Essas validações devem levar em consideração as restrições existentes nas políticas e/ou processos da organização a serem adotados no repositório. Por exemplo, o *processo de certificação* utilizado pode determinar que o conteúdo de uma versão de um ativo não seja alterado caso a versão se encontre em avaliação. Podem existir também, políticas internas que determinam, por exemplo, o formato do rótulo de uma versão (*version label*). Por conta disto, o sub-módulo *VersionManager* disponibiliza a interface *IVersionPolicy*, que deve ser usada quando se deseja customizar as políticas de versionamento adotadas no repositório. Esta interface possui serviços que informam se uma versão pode ser excluída, se o rótulo da versão é válido, se uma versão pode ser alterada, entre outros.

Adicionalmente, o *VersionManager* possui duas outras interfaces: 1 – *IVersionDiff* que possui operações para verificar diferenças entre versões; e 2 – *VersionHistory* que possui serviços para analisar o histórico de alterações das versões de um ativo. Essas duas interfaces não foram contempladas na implementação inicial dessa arquitetura.

Dependency Manager. De acordo com o requisito #9, os produtores de ativos devem ser capazes de informar possíveis dependências dos seus ativos com outros. Para isso, foi criado o sub-módulo *DependencyManager* que oferece os serviços necessários para o gerenciamento de relações entre ativos. As relações informadas devem englobar dependências com ativos internos (armazenados no repositório) ou externos (localizados em outras fontes). Segundo [Apperly, 2002], essas relações devem suportar um esquema de classificação flexível que permita definir diferentes tipos de relacionamentos, tais como, "*é parte de*", "*é um*", "*usa*", "*é composto por*".

Os serviços fornecidos pelo *DependencyManager* também englobam funções que servem para auxiliar o *AssetRegister* na validação de consistência de operações que envolvem a remoção de um ativo. Um exemplo muito comum de restrição é impedir que um ativo seja removido caso outro ativo do repositório seja dependente dele. Uma solução simples para esse caso é suportar a *remoção lógica* do ativo, dessa forma ele continuaria armazenado no repositório, porém, seria marcado como depreciado (*deprecated*).

AssetExtractor. Segundo Mascena [Mascena, 2006], além da inserção manual de ativos, um ambiente focado em reuso de software, como é o caso do repositório proposto, idealmente deve suportar um processo automático de recuperação de conteúdo legado. Este processo é efetuado pelo sub-módulo *AssetExtractor* previsto na arquitetura. O *AssetExtractor* é responsável por incrementalmente e continuamente monitorar as atividades de produção de ativos na organização e torná-los disponíveis no repositório. Para isso, ele acessa as diferentes fontes de informação existentes na organização e extrai ativos que possam ser reusados. Os

ativos extraídos devem passar por um processo automático de análise e classificação antes de serem finalmente armazenados no repositório. É importante observar que o *AssetExtractor* aparece pontilhado na Figura 3.54, pois representa um módulo planejado da arquitetura que não foi implementado na versão inicial do sistema.

Uma vez que os ativos foram produzidos e armazenados no repositório, o passo seguinte é analisarmos os componentes da arquitetura que lidam com o processo de consumo desses ativos.

3.2.4 Modulo de Consumo

O módulo de Consumo (*Consumption*) é o responsável pela interação dos consumidores de ativos com o sistema. Ele inclui os serviços de busca e recuperação de ativos, cadastro de interesse, gerenciamento de mudanças e *feedback* do usuário. Esses serviços estão distribuídos pelos 5 (cinco) sub-módulos que o constituem: Asset*Searcher*, *Retriever*, *FeedbackManager*, *Change Manager* e *Interest Manager*. A Figura 3.5 mostra como estes elementos estão organizados.

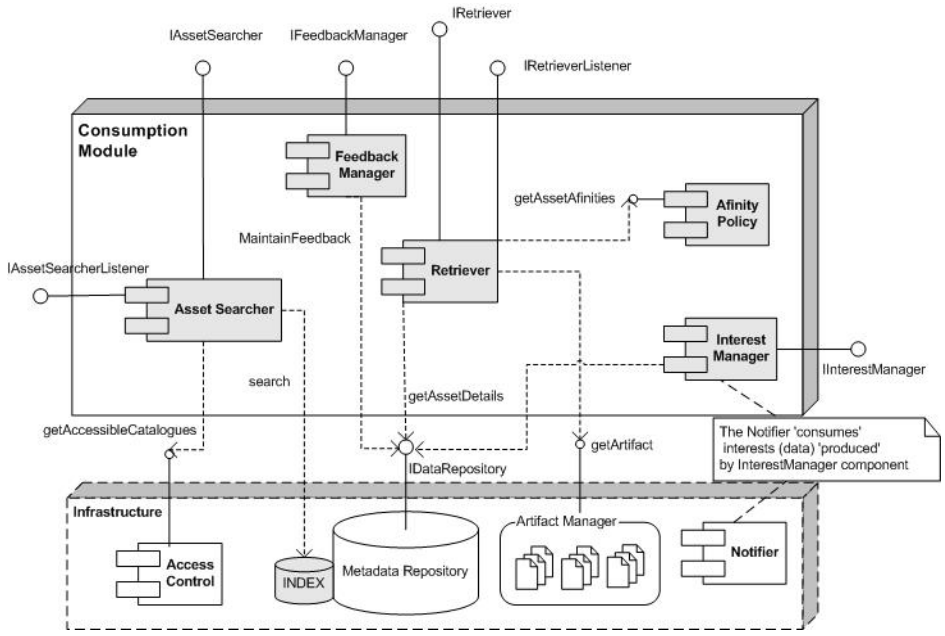


Figura 3.5 – Módulo de Consumo

AssetSearcher. Este sub-módulo é o responsável pela interpretação das consultas enviadas ao repositório e recuperação das informações dos ativos relevantes, ou seja, que "casam" com as consultas recebidas. Para isso, o *AssetSearcher* utiliza a base indexada (*Index*) mantida pelo componente de indexação do módulo de infra-estrutura.

A interface *IAssetSearcher*, disponibilizada por este sub-módulo, suporta os tipos de busca citados no requisito #4. Estes tipos incluem: busca em texto livre, busca com palavra-chave e busca por classificadores. O retorno desses serviços é uma coleção de objetos contendo as informações básicas que descrevem os ativos retornados na busca.

É importante ressaltar, que o escopo da busca pode variar de acordo com o conjunto de catálogos de ativos acessíveis pelo usuário que realizou a consulta. Essa informações é obtida através do módulo *AccessControl* e é utilizada como um filtro na operação que realiza a busca no índice.

Em um repositório de reuso, a monitoração do serviço de busca pode trazer benefícios para o processo de análise das atividades de reuso na organização. Um exemplo simples é utilizar o histórico de buscas mal sucedidas (que não retornaram resultados) para identificar possíveis "demandas reprimidas" no repositório e, conseqüentemente, servir como indicador de prováveis componentes a serem criados. Para permitir esse tipo de monitoração, o módulo *AssetSearcher* disponibiliza a interface *IAssetSearcherListener* cujos métodos são chamados nos principais estágios do processo de busca.

Retriever. A recuperação dos ativos é uma funcionalidade fundamental em qualquer repositório de reuso. Na arquitetura proposta, o módulo responsável por esta funcionalidade é o *Retriever*. Este módulo prove a interface *IRetriever* que permite a recuperação dos ativos através do *Metadata Repository* e *Artifact Manager*.

Durante qualquer processo de busca é muito comum o usuário querer visualizar os detalhes dos elementos que mais lhe interessaram. Com o intuito de promover o reuso, a visualização dos detalhes de um ativo pode conter uma lista

sugestiva de ativos similares ou relacionados ao ativo visualizado. Essa estratégia de sugestão pode ser comumente identificada em *sites* de venda de produtos. Nestes *sites*, a tela de detalhes de um produto é apresentada junto com uma lista de recomendação de outros produtos, que foram comprados pela maioria dos usuários que se interessaram pelo produto visualizado.

Para suportar, de maneira flexível, o uso desses mecanismos de recomendação, foi definida a interface *IAfinityPolicy*. Tal interface possui um serviço, que retorna uma lista de recomendações relacionadas a um determinado ativo. A política utilizada para gerar as recomendações pode ser alterada facilmente através de implementações distintas desta interface.

Utilizando a mesma abordagem da maioria dos módulos, o *Retriever* também possui uma interface cujos serviços são chamados durante o processo de recuperação de componentes. As implementações dessa interface podem acionar eventos como o registro de *downloads* efetuados pelos usuários. Estes registros servirão, por exemplo, como base para a geração de diversos tipos de relatórios voltados para a análise de consumo dos ativos do repositório.

Feedback Manager. Este sub-módulo trata as operações relacionadas à manutenção de *feedback* dos consumidores a respeito dos ativos que usaram o mesmo (requisito #10). O conjunto de *feedbacks* serve como um histórico de uso dos ativos e pode indicar o nível de aceitação e eficiência dos mesmos. A estrutura de cada *feedback* deve incluir informações como o contexto de utilização, o resultado do uso (sucesso ou falha) e métricas que apresentem informações a respeito do esforço requerido para reusar o ativo.

Change Manager. Este sub-módulo é responsável pela manutenção das solicitações de mudanças dos ativos. Opcionalmente, este sub-módulo pode ser substituído ou integrado a uma ferramenta de controle de mudanças já existente. A interface implementada por ele (*IChangeManager)* é bem simples e inclui basicamente as operações de solicitar e resolver mudanças/*bugs*.

Interest Manager. Permite aos usuários configurar seus interesses em receber notificações do repositório. Estes interesses podem variar desde interesse em um ativo específico, até interesses mais gerais em um determinado domínio de componentes. Algumas considerações a respeito dos tipos de interesse serão feitas quando forem discutidas as lições aprendidas com a implementação do repositório.

3.2.5 Módulo de Gerenciamento

Para permitir o monitoramento das atividades de reuso suportadas pelo repositório e garantir a qualidade dos ativos disponibilizados foi definido o módulo de gerenciamento. Este módulo manipula informações armazenadas no *Metadata Repository* e disponibiliza serviços para os usuários com perfil de administrador e certificador. Seus principais sub-módulos são: Asset *Certifier, Advertisement, Reports, Catalogue Manager* e *Administration Services.* Figura 3.6 ilustra a organização interna do módulo de gerenciamento. Em seguida, a responsabilidade de cada um dos seus sub-módulos é descrita.

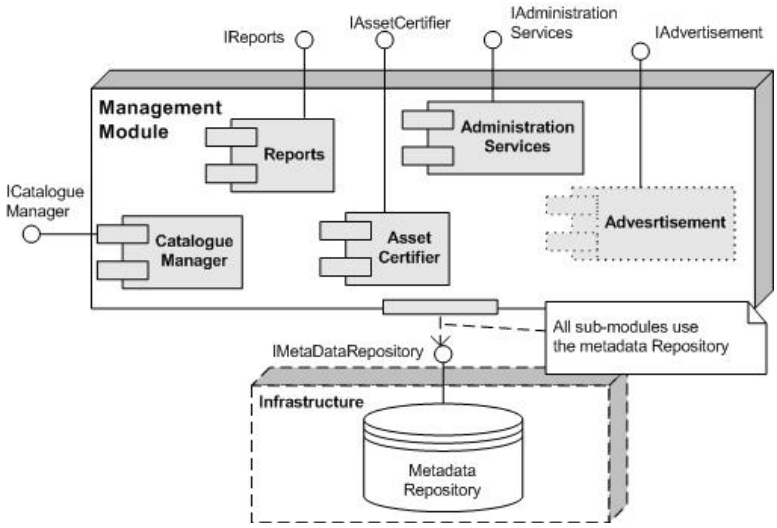


Figura 3.6 – Módulo de Gerenciamento

Asset Certifier. Permite que os certificadores garantam a qualidade dos ativos disponíveis para consumo. A certificação consiste em avaliar o ativo e garantir a sua qualidade conforme um *processo de certificação* a ser seguido (ver Apêndice

C). O resultado da avaliação é descrito através de um documento que apresenta maiores detalhes sobre o processo realizado. Esse resultado é anexado ao ativo no repositório e um *status* de qualidade é atribuído ao mesmo, dessa forma, é possível, por exemplo, realizar uma busca informando que apenas ativos certificados sejam considerados.

Advertisement. É um sub-módulo simples que funciona como um gerenciador de notícias. Seus serviços permitem a manutenção de anúncios destinados a atrair a atenção dos usuários e promover a cultura de reuso na organização. Este sub-módulo encontra-se pontilhado na Figura 3.5, pois não foi contemplado na versão implementada inicialmente.

Reports. Este sub-módulo é responsável pela geração dos relatórios a serem usados pelos gerentes da organização. Ele permite que novos relatórios sejam adicionados de forma flexível e deve suportar diferentes formatos de saída, tais como, PDF, html, entre outros. O Apêndice B apresenta alguns exemplos de relatórios que podem ser gerados por este sub-módulo.

Catalogue manager. Este sub-módulo é responsável pela manutenção da hierarquia de catálogos de ativos. É neste momento que os catálogos são criados e um valor semântico para eles é fornecido, ou seja, é decidido se um catálogo representará um departamento, ou uma equipe de projeto, ou uma unidade de negócio, ou ainda um domínio de aplicação. Além disso, a manutenção dos catálogos também envolve a especificação das permissões que determinarão como será o acesso aos mesmos.

Administration Services. Este sub-módulo agrupa uma série de gerenciadores responsáveis pela manutenção do domínio de valores de diversas entidades utilizadas no repositório. Estas entidades incluem: os tipos de artefatos, tipos de ativos, facetas, entre outros. Também faz parte desse sub-módulo, o gerenciador de

usuários (*UserManager*) que é responsável pela manutenção de usuários do repositório.

3.2.6 Resumo da arquitetura

Como pode ser observado na especificação da arquitetura, todos os módulos e sub-módulos do repositório externam suas funcionalidades através de interfaces, como por exemplo: *IAssetProduction, IAssetSearcher, IReports*, entre outras. Além disso, os pontos de extensão (ex.: *IUserManagerAdapter*) e todas as comunicações existentes entre os componentes e sub-módulos internos também são feitas via interfaces. Essa característica existe porque o conceito central da arquitetura proposta é criar uma API de serviços que possam ter suas implementações facilmente substituídas, configuradas ou adaptadas de acordo com as necessidades do contexto onde o repositório será implantado.

A Seção seguinte relata a experiência de desenvolvimento de um repositório que utilizou a arquitetura descrita nesta Seção.

3.3 Experiência de implementação

Uma implementação inicial baseada na solução proposta por esta dissertação foi desenvolvida em conjunto com o Centro de Estudos e Sistemas Avançados do Recife (C.E.S.A.R.) e teve o apoio (financiamento) do governo federal, representado pela FINEP[9], a qual publicou - no Diário Oficial da União de 31/08/2004 - uma seleção pública de propostas para apoio à cooperação tecnológica entre o setor produtivo e instituições científicas e tecnológicas (CHAMADA PÚBLICA MCT/FINEP/Ação Transversal-Cooperação ICTs-Empresas-02/2004 - Referência FINEP n.º 2176/04).

O desenvolvimento do repositório proposto fez parte da principal meta do projeto industrial submetido à FINEP (GComp- Gerenciador de Componentes). Tal projeto envolveu a criação de processos, métodos e ferramentas para suportar as

[9] http://www.finep.gov.br/

atividades de reuso de software em empresas. Esta Seção apresenta a experiência de implementação da primeira versão deste produto, abordando o processo de desenvolvimento utilizado, as tecnologias e *frameworks* adotados na solução, os resultados preliminares e os problemas encontrados durante o desenvolvimento.

3.3.1 Processo de desenvolvimento

O processo de desenvolvimento adotado foi fortemente baseado no RUP - Rational Unified Process [RUP, 2006] - um processo de engenharia de software cujas principais características são um desenvolvimento iterativo e incremental, orientado a objetos, com foco na criação de uma arquitetura robusta, análise de riscos e utilização de casos de uso para o desenvolvimento. O RUP foi desenvolvido para ser aplicável a uma grande classe de projetos diferentes e pode ser considerado como um *framework* genérico para processos de desenvolvimento. Isso significa que ele deve ser configurado para se adequar à realidade da organização.

No caso especifico deste trabalho, um conjunto de artefatos do RUP foi considerado e alguns *templates* foram customizados. Além disso, por se tratar de um projeto de inovação sendo executado com a parceria entre academia e indústria, foram adicionadas duas importantes atividades no ciclo de desenvolvimento do processo: (1) *Atividade de pesquisa* e (2) *Avaliação de arquitetura*.

- **Atividade de pesquisa**: corresponde aos diversos estudos que foram realizados no decorrer do projeto. Estes estudos serviram de base para a atividade de especificação de requisitos e tinham como principal objetivo a identificação e análise das soluções existentes para os problemas que seriam tratados. A atividade de pesquisa foi totalmente incorporada ao processo e exigiu planejamento e acompanhamento por parte do gerente de projeto.
- **Avaliação de arquitetura**: esta atividade englobou a aplicação de um método baseado em cenários para avaliar a qualidade do projeto arquitetural do repositório, antes de sua implementação. Os resultados obtidos com essa atividade comprovaram que a aplicação de tais métodos no contexto industrial pode contribuir, de maneira significativa, para o

aumento da qualidade dos produtos avaliados. O Capítulo 4 apresenta com mais detalhes a avaliação realizada na arquitetura do repositório proposto. Mais informações a respeito do experimento realizado também podem ser encontradas em [Burégio, 2006].

Para termos uma idéia de onde essas atividades foram encaixadas no processo, a Figura 3.7 apresenta o ciclo iterativo de desenvolvimento do RUP acrescido das atividades de pesquisa e avaliação de arquitetura. Por questão de escopo, as disciplinas do RUP apresentadas na Figura não serão descritas aqui.

Figura 3.7 – Processo de desenvolvimento adotado

Equipe de projeto

A equipe de projeto foi formada por pesquisadores do grupo RiSE[10], juntamente com integrantes da indústria. O time foi composto por 1 consultor de reuso (líder de pesquisa), 1 arquiteto de software (líder técnico), 1 analista de sistemas (líder de equipe), 3 engenheiros de software, e 1 gerente de projeto. Além disso, existiam algumas pessoas que eventualmente interagiam com o projeto, tais como, gerente, engenheiro de configuração (CM), SQA (*Software Quality Assurer*) e um comitê formado por arquitetos seniores do C.E.S.A.R que participaram de algumas decisões técnicas.

[10] http://www.rise.com.br

As principais tecnologias e *frameworks* utilizados na implementação do repositório são mostradas na próxima Seção.

3.3.2 Tecnologias e *frameworks*

A implementação do repositório consistiu em uma aplicação web implementada com a linguagem Java[11]. A codificação seguiu o padrão J2EE[12] e os padrões de codificação da SUN[13].

A seguir serão discutidas as principais tecnologias e *frameworks* adotados na implementação do repositório, a saber: (1) Biblioteca de recuperação de informação, (2) Mapeamento Objeto/relacional, (3) Framework para aplicação Web e (4) Container para Inversão de Controle (IoC).

1. Biblioteca de recuperação de informação

Como visto no Capítulo 2, busca e recuperação de componentes é tradicionalmente o tópico de pesquisa mais explorado nos trabalhos acadêmicos envolvendo sistemas de repositórios de reuso [Mili et al., 1998]. Estes trabalhos, na sua grande maioria, envolvem questões relacionadas ao processo de recuperação de informação (*information retrieval*) em tais repositórios.

De uma maneira geral, um processo de recuperação de informação leva em consideração o conceito de *relevância* de um documento no contexto de uma determinada consulta, ao contrário da *recuperação de dados* (*data retrieval*), que está interessada em recuperar todos os dados que casam com uma consulta específica, geralmente expressa como uma *expressão regular* ou com uma *expressão da álgebra relacional* [Baeza-Yates & Ribeiro-Neto, 1999].

Na arquitetura proposta, dois sub-módulos lidam com partes distintas do processo de recuperação de informação: o sub-módulo *Indexer* do módulo de *Infra-estrutura* e o *AssetSearcher* pertencente ao módulo de *Consumo*. Para a implementação deles, foram analisadas algumas bibliotecas de código aberto que

[11] http://www.java.sun.com
[12] http://www.corej2eepatterns.com
[13] http://java.sun.com/blueprints/code/projectconventions.html

implementam o conceito de recuperação de informação. Dentre estas bibliotecas pode-se destacar duas: *Egothor* e *Lucene* [Gospodnetic & Hatcher, 2004].

A *Egothor*[14] é uma biblioteca de classes Java de código fonte livre e aberto. Sua distribuição vem com uma série de aplicações prontas para uso, tais como um *web crawler* (*Capek*), um indexador de arquivos com interface gráfica *Swing*, entre outros. Ele também prove *parsers* para diversos formatos de arquivos, tais como *PDF* e documentos do *Microsoft Word*.

O projeto *Lucene*[15] é uma biblioteca de código fonte livre e aberto formada por classes Java, mas possui portabilidade para outras linguagens, como C++ e C#. Seus algoritmos internos são muito similares aos usados pelo *Egothor*. Ela também possui suporte a vários formatos de arquivos e possui uma arquitetura bem projetada com vários pontos de extensão. O *Lucene* é comparável ao *Egothor* em muitos aspectos e são duas boas opções a serem consideradas na implementação de um mecanismo de busca em repositórios de reuso.

Para a solução, foi escolhido o *Lucene*, pois além de possuir uma boa documentação, ele apresenta um modelo bastante flexível. O modelo do *Lucene* define a interface *Document* que representa a abstração do conteúdo de um documento a ser indexado. Um *Document* é composto por um conjunto especificado e customizado de campos (*fields*). As consultas podem ser efetuadas tanto em campos simples, quanto em múltiplos campos. Os campos, por sua vez, podem ser de vários tipos, os quais determinam a maneira como os mecanismos de busca e indexação irão manipulá-los. A partir deste modelo flexível, estratégias complexas de busca e recuperação de informação podem ser implementadas.

2. Mapeamento Objeto/relacional

Como visto na Seção 3.2.2, o componente *MetadaRepository* do módulo de infra-estrutura representa uma abstração do mecanismo de persistência a ser utilizado no repositório. Sua implementação utilizou os conceitos de mapeamento objeto/relacional.

[14] http://www.egothor.org/
[15] http://lucene.apache.org

O mapeamento objeto/relacional (ORM) consiste na persistência automatizada de objetos de uma linguagem O.O. em tabelas de um banco de dados relacional. Para isto, são utilizados metadados que descrevem o mapeamento entre os objetos e a base de dados. Esta abordagem, em sua essência, trabalha com a transformação (reversível) de dados de uma representação para outra e resolve os problemas de armazenamento por providenciar um alto nível de abstração, comumente chamado de persistência transparente (*transparent persistence*).

Os primeiros trabalhos envolvendo técnicas de mapeamento objeto/relacional surgiram no final da década de 80 [Bauer & King, 2005] e até hoje vem desempenhando um importante papel no desenvolvimento da camada de persistência de aplicações que utilizam linguagens orientadas a objetos como Java, por exemplo.

A grande vantagem de se utilizar essa abordagem está relacionada ao aumento de produtividade. Pois, com o uso do mapeamento O/R o desenvolvedor lida apenas com objetos e com as regras de negócio da aplicação, não precisando, portanto, se preocupar com a especificação de comandos em outras linguagens de consulta a dados como SQL, que geralmente é uma atividade repetitiva e muitas vezes demorada. Isto certamente pode implicar em perda de *performance*. Porém, se o ORM for implementado como um *middleware*, muitas soluções de otimização podem ser utilizadas, o uso de *cache* é um exemplo típico destas otimizações.

Em Java, existem alguns *frameworks* de código aberto que implementam o mapeamento objeto-relacional. Dentre as opções existentes, a mais expressiva atualmente é o *Hibernate*[16] [Bauer & King, 2005]. O *Hibernate* tem se tornado um padrão no desenvolvimento de aplicações em Java que utilizam ORM. Ele possui compatibilidade com a maioria dos servidores de banco de dados existentes no mercado, o que conseqüentemente confere um certo nível de portabilidade para as aplicações que o utiliza. Assim como o *Lucene*, sua arquitetura possui variados pontos de extensão. Além disso, sua distribuição prove a implementação de diversos serviços como, por exemplo, *cache* de objetos e mecanismos que permitem configurar consultas sob demanda.

[16] http://www.hibernate.org

A implementação do componente *MetadaRepository* do módulo de infra-estrutura foi essencialmente realizada sobre a arquitetura do *Hibernate* e sua portabilidade foi testada em dois servidores de banco de dados, a saber: *Oracle*[17] (servidor comercial) e *PosrtgreSQL*[18] (solução *open-source*).

3. Framework para aplicação Web

O padrão de projeto MVC (Model-View-Controller) é comumente utilizado em aplicações que possuem uma interface gráfica com o usuário (GUI). Porém, o uso deste padrão encontra dificuldades quando a aplicação em questão é um sistema web, como é o caso da primeira versão do repositório desenvolvida. A principal dificuldade é a implementação do elemento *Controller*, pois, nestes sistemas, os clientes (*View*) não mantêm uma conexão aberta com o servidor, o que dificulta o processo de notificação dos mesmos quando mudanças acontecem nos dados da aplicação (*Model*).

Para minimizar esse problema, vários *frameworks* para aplicações web, em Java (*Cocoon*, *SpringMVC*, *Struts*, *Tapestry*, *WebWork2*), foram criados nos últimos anos, com o objetivo de desempenhar o papel do elemento *Controller* existente no padrão MVC.

Estes *frameworks* permitem a separação das camadas de dados, controle e visualização. Esta separação oferece vantagens para desenvolvedores, como a otimização das habilidades de equipes e a redução de custos associados ao desenvolvimento, além de favorecer a extensibilidade e a reutilização do código

O *framework* adotado nesta versão inicial do repositório foi o *Struts* [Husted et al., 2003]. Apesar de existirem outras abordagens possíveis para processamento de requisições de clientes, o *Apache Struts* tem se tornado uma unanimidade do desenvolvimento Java para Web [Husted et al., 2003]. O *Struts* possui ainda uma boa separação de conceitos, é muito bem conhecido entre os desenvolvedores e suportado por muitas ferramentas de desenvolvimento. Além

[17] http://www.oracle.com
[18] http://www.postgresql.org

disso, ele possui mecanismos que facilitam a implementação de interfaces internacionalizáveis e com o *layout* flexível.

4. Container de Inversão de Controle (IoC)

Como dito na Seção 3.2.6, o conceito central da arquitetura proposta é criar uma API de serviços os quais possam ter suas implementações substituídas, configuradas ou adaptadas de acordo com as necessidades do contexto onde o repositório será implantado.

Para facilitar essa customização e substituição das implementações dos serviços, o repositório construído fez uso da Inversão de Controle (*Inversion of Control - IoC*). A Inversão de Controle é um importante princípio que pode ser utilizado para reduzir o acoplamento entre os componentes de um sistema computacional. Com ela, os componentes da aplicação não usam diretamente outros componentes, pois as instâncias dos objetos que implementam uma determinada interface são "injetadas" automaticamente. Por esta razão, este conceito vem recentemente sendo chamado, por alguns pesquisadores, de injeção de dependência (*Dependency Injection*) [Fowler, 2004].

Existem alguns *containers* em Java (*PicoContainer*[19], *Spring*[20], *Hivemind*[21]) que podem ser usados para injetar dependências automaticamente. Eles permitem que os serviços requeridos pelos componentes sejam configurados e, a partir daí, o *container* é o responsável por criar os objetos, juntá-los através da "injeção" das propriedades necessárias, e ainda determinar quando os serviços serão invocados.

O *Hivemind* foi o *container* utilizado na implementação do repositório. Ele possui um foco maior no conceito de IoC e implementa os 3 tipos injeção de dependência definidos em [Fowler, 2004]. Com ele, toda a aplicação pode ser vista como um conjunto de pontos de serviços (*service-points*) que podem ser facilmente combinados. Desta forma, todas as principais interfaces previstas na arquitetura foram mapeadas para *service points* do *Hivemind* e, conseqüentemente as

[19] http://www.picocontainer.org/
[20] http://www.javafree.org/wiki/Spring
[21] http://jakarta.apache.org/hivemind/

implementações destas interfaces podem ser alteradas facilmente sem mudança alguma no sistema.

Além das soluções apresentadas, outras bibliotecas disponíveis e de uso gratuito também foram reutilizadas na implementação, tais como: *Log4J* - usado para o log de erros e de execução do repositório, *Jasper* – biblioteca para geração de relatórios em diversos formatos, e *Quartz* – biblioteca usada para criação do componente *Scheduler* utilizado para agendar alguns eventos disparados automaticamente pelo repositório, como é o caso de algumas notificações aos usuários.

3.3.3 Resultados do projeto

Esta Seção apresenta os resultados obtidos com a execução do projeto, que teve duração de 12 meses. É importante ressaltar que o projeto envolveu outras metas além do desenvolvimento do repositório. Logo, os resultados apresentados aqui se referem apenas à meta relacionada com a construção do repositório de reuso.

Esforço de desenvolvimento

O tempo gasto em cada atividade do projeto era reportado diariamente pelos membros da equipe, através de um sistema interno utilizado para coleta de métricas. Os dados colhidos eram avaliados mensalmente pelo SQA, alocado no projeto, e servia de *feedback* para os novos planejamentos de atividades a serem realizados pelo gerente de projeto. A Figura 3.8 apresenta um gráfico com o resultado total de horas acumuladas em atividades relacionadas às seguintes fases do desenvolvimento do repositório: Pesquisa, Requisitos (Req.), Análise e Projeto (A&P), Implementação (Imple), Testes e Implantação (Impla). O total de horas do projeto (incluindo todas as metas) foi de 12.960 horas.

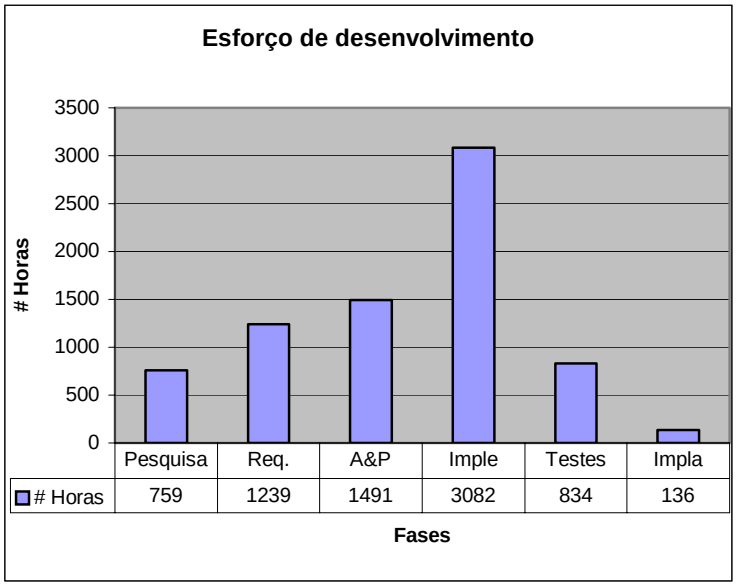


Figura 3.8 – Esforço de desenvolvimento

Produto obtido

O principal resultado obtido com o projeto foi um produto comercial que está sendo diretamente aplicado a fábricas de softwares brasileiras. A primeira versão do repositório foi implementada em 423 classes, divididas em 83 pacotes e com 19.367 linhas de código não comentadas. Esta versão do produto já se encontra em produção em um contexto com mais de 1.500 desenvolvedores acessando diretamente o repositório corporativo. A Figura 3.9 apresenta a tela inicial da ferramenta.

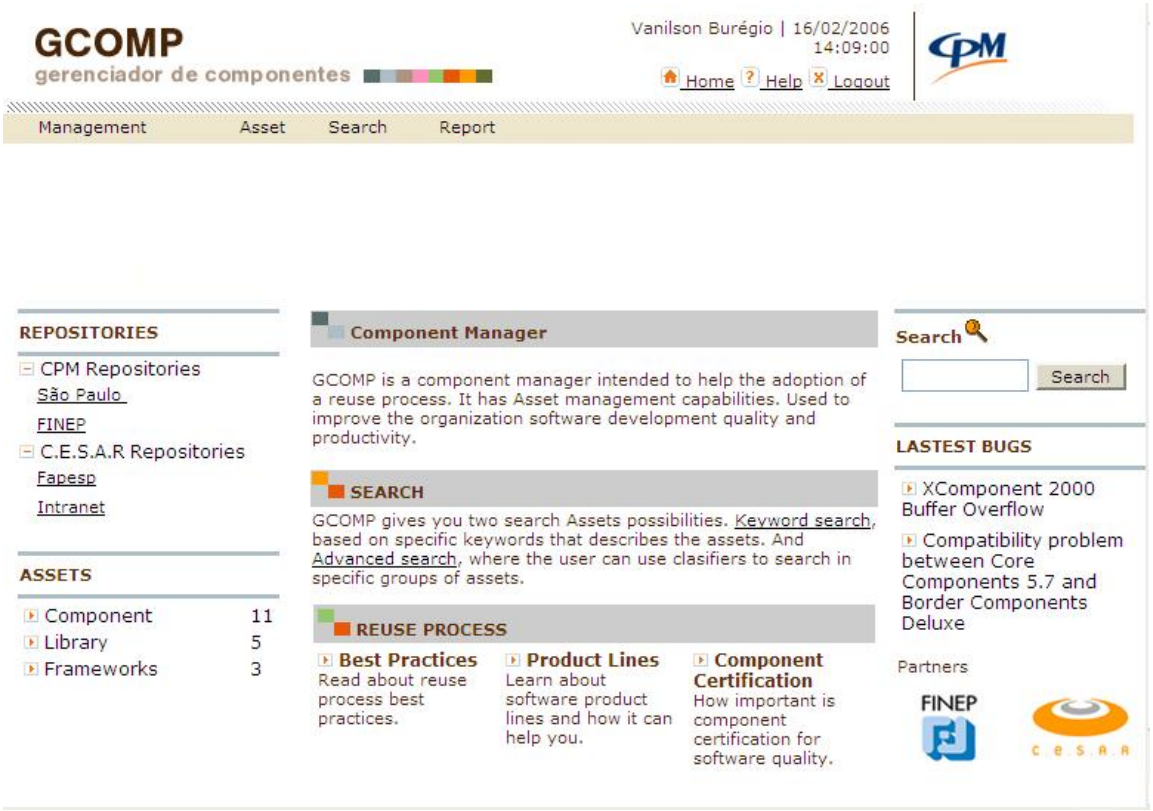


Figura 3.9 - Tela inicial do repositório

Os bons resultados obtidos com a implantação do repositório geraram uma extensão do projeto por mais 6 meses. Atualmente, está sendo feito um estudo para o acréscimo de novas funcionalidades à ferramenta. Além disso, já está em desenvolvimento um *plug-in* que permitirá aos usuários (principalmente desenvolvedores) acessar as funcionalidades do repositório através da IDE do Eclipse[22]. A Figura 3.10 apresenta a tela do protótipo do *plug-in* implementado. Como pode ser visto ele é formado por 3 *views* principais: (1) *Browsing* – permite navegar pelos catálogos de ativos do repositório, (2) *Search Viewer* – permite a realização de buscas no repositório, e (3) *Asset details Viewer* – responsável pela visualização das informações do ativo juntamente com os *links* para os arquivos que compõem o seu conteúdo (número 4 na Figura).

[22] http://www.eclipse.org

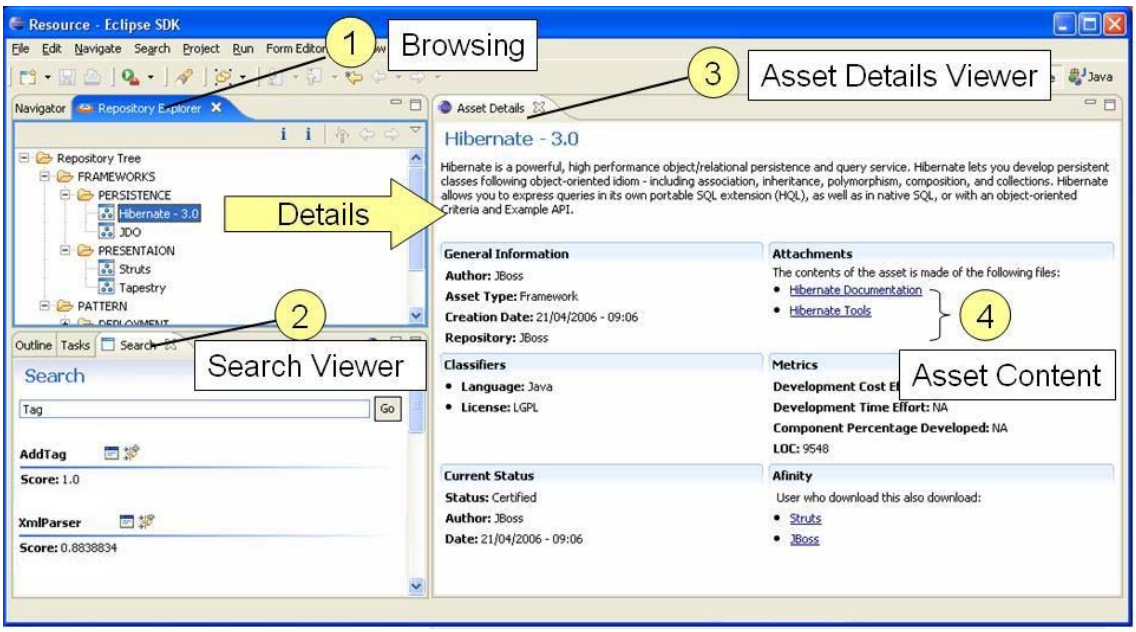


Figura 3.10 – Tela do protótipo de *plug-in* para Eclipse

3.4 Resumo do capítulo

Este Capítulo apresentou os principais aspectos do repositório proposto. Os requisitos do repositório foram inicialmente definidos e, em seguida, as responsabilidades dos principais módulos que compõem sua arquitetura foram descritas e a estrutura interna desses módulos foi apresentada. Finalmente, a experiência de implementação de um produto comercial, baseado nas especificações propostas, foi relatada.

O próximo Capítulo apresenta alguns exemplos de utilização do produto através de cenários e em seguida será apresentado o processo de avaliação de arquitetura realizado, conforme descrito na Seção 3.3.1.

4 Análise do Repositório

Como visto no Capítulo anterior, uma implementação inicial baseada na solução proposta por esta dissertação foi desenvolvida e o principal resultado obtido foi um produto comercial utilizado em fábricas de software brasileiras. Logo, visando um melhor entendimento de como este produto satisfaz os principais interesses de seus usuários, este Capítulo apresenta alguns cenários de uso do mesmo sob diferentes pontos de vista.

Após isso, será apresentado o processo de avaliação de arquitetura realizado durante o desenvolvimento do repositório (antes de sua implementação). Por fim, alguns comentários a respeito das principais questões que surgiram durante o desenvolvimento serão apresentados.

O restante deste Capítulo está organizado da seguinte forma: a **Seção 4.1** apresenta os exemplos de utilização do repositório. A **Seção 4.2** relata a experiência realizada de avaliação da arquitetura. A **Seção 4.3** relata os principais problemas e questões resolvidas durante o desenvolvimento do produto. E, finalmente, a **Seção 4.4** resume o capítulo.

4.1 Exemplos de Utilização

Para ilustrar o funcionamento do repositório obtido, esta Seção apresenta como alguns cenários típicos de utilização do repositório são executados pelos usuários de cada módulo do repositório: produção, consumo e gerenciamento.

Como o repositório inicialmente implementado foi um sistema web, a execução dos cenários de uso será apresenta através do fluxo navegacional das telas do sistema. Para cada cenário será apresentada uma descrição, o fluxo navegacional das telas envolvidas e, em alguns casos, os detalhes do conteúdo das principais seções de algumas telas. Os cenários são os seguintes:

1. Módulo de Produção: Publicar nova versão de um ativo;
2. Módulo de Consumo: Recuperar ativo;
3. Módulo de Gerenciamento: Certificar ativo;

4.1.1 Cenário 1: publicar nova versão de um ativo

Um usuário com perfil de produtor deseja disponibilizar para todos os usuários do repositório a nova versão do seu componente de controle de acesso que foi construída em Java. O componente possui 5.000 linhas não comentadas de código e foi desenvolvido em 160 horas de trabalho. Os artefatos que ele deseja publicar são: documentação do usuário, código fonte e diagrama de classes.

Fluxo Navegacional

A Figura 4.1 apresenta o fluxo navegacional das telas do repositório envolvidas para a realização desse cenário, em seguida, os passos executados serão detalhados.

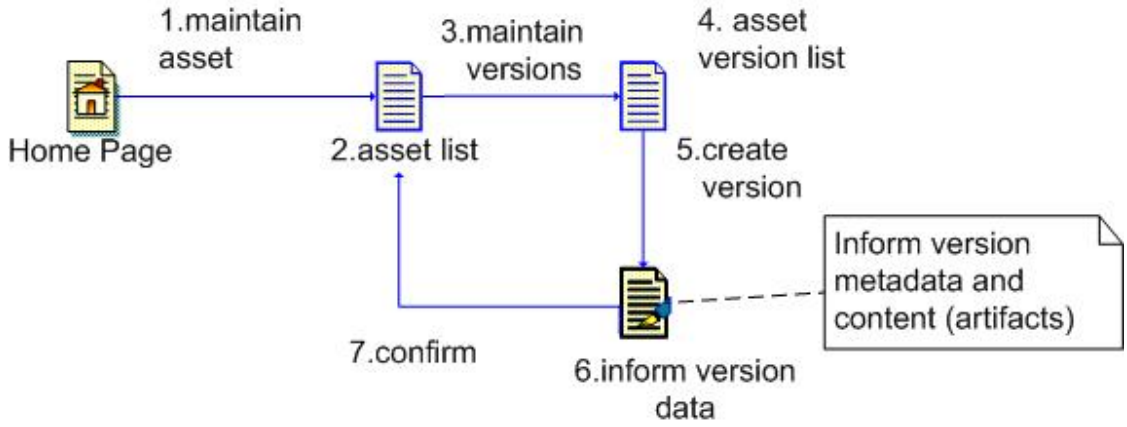


Figura 4.1- Fluxo navegacional : Criação de versão

1. Após ter logado no repositório, o usuário produtor acessa a opção de manter ativos, presente na tela principal;
2. O sistema apresenta uma tela similar à mostrada na Figura 4.2, onde o usuário pode informar algumas opções de filtro (Figura 4.2-A) para a listagem dos ativos que podem ser modificados por ele;

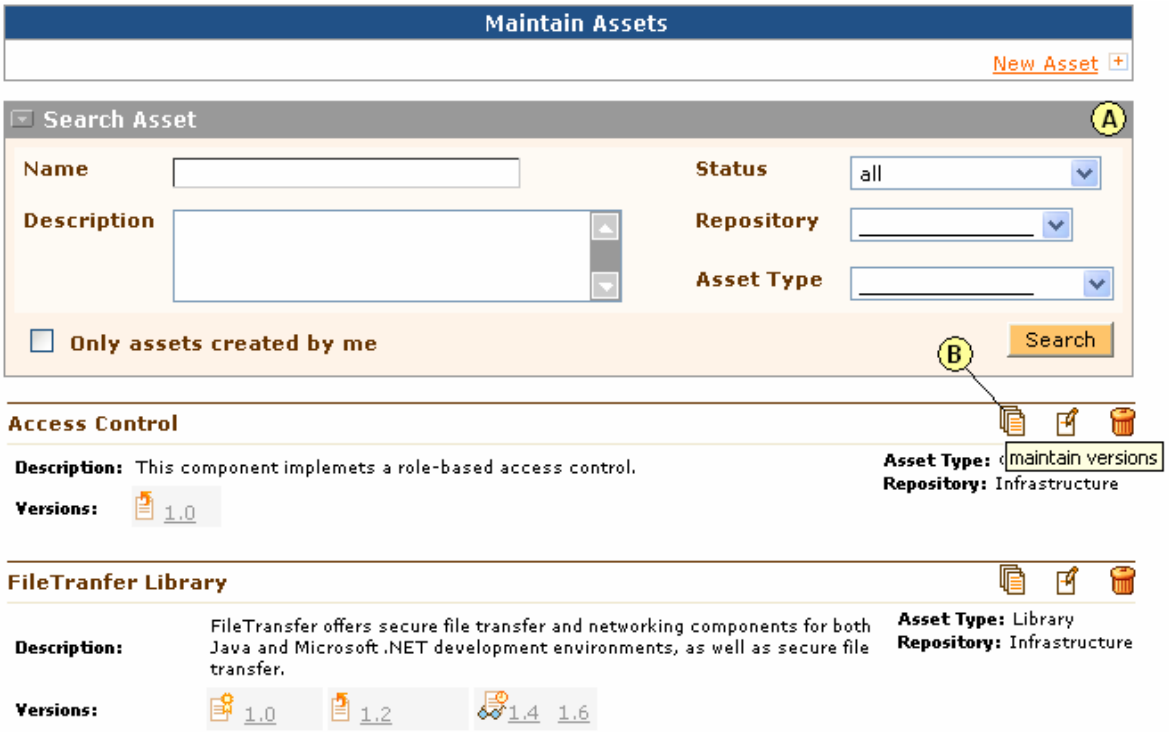


Figura 4.2 - Tela de manutenção de ativos

3. O usuário então escolhe o ativo de controle de acesso e seleciona a opção de manter versões para este ativo (Figura 4.2-B);
4. O sistema apresenta uma tela com a lista de versões do ativo selecionado, neste caso o controle de acesso possui apenas uma versão;
5. O usuário seleciona a opção de criar versão;
6. O sistema apresenta uma tela com as informações da nova versão do ativo a serem preenchidas pelo usuário, estas informações incluem: descrição, rótulo da versão, classificadores, métricas, artefatos, dependências, entre outras. A Figura 4.3 apresenta a primeira parte da seqüência de telas envolvidas na criação de uma versão.

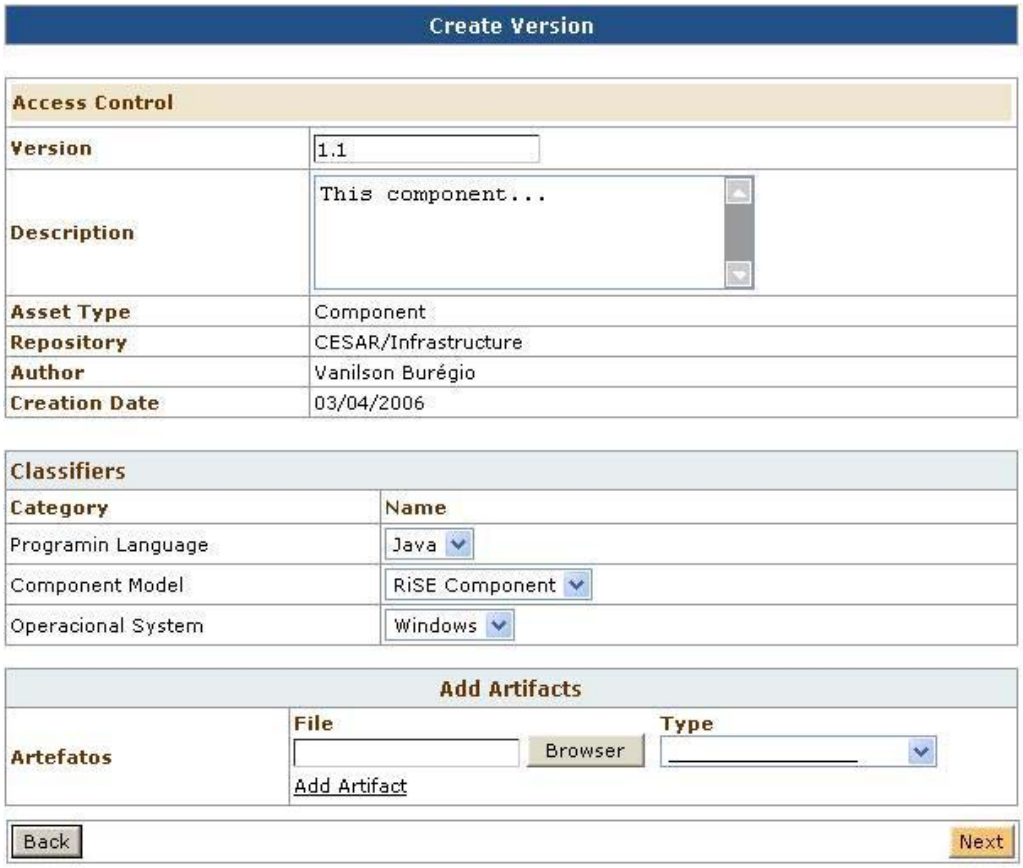


Figura 4.3 – Tela de criação de versão

7. O usuário informa os metadados da nova versão, faz o *upload* dos seus artefatos e confirma a operação;
8. O sistema salva as informações da nova versão do ativo e volta para a tela anterior com a listagem dos ativos do usuário.

4.1.2 Cenário 2: recuperar ativo

Um desenvolvedor deseja encontrar uma biblioteca que efetue operações de transferência de arquivos e que possa ser utilizada em uma aplicação Java. Como será preciso transferir dados com segurança, é muito importante que a biblioteca reutilizada tenha qualidade comprovada. Para isso, ele utilizará os serviços oferecidos pelo módulo de consumo do repositório.

Fluxo Navegacional

A Figura 4.4 apresenta o fluxo de navegação a ser executado por um usuário que deseja buscar e recuperar um ativo no repositório.

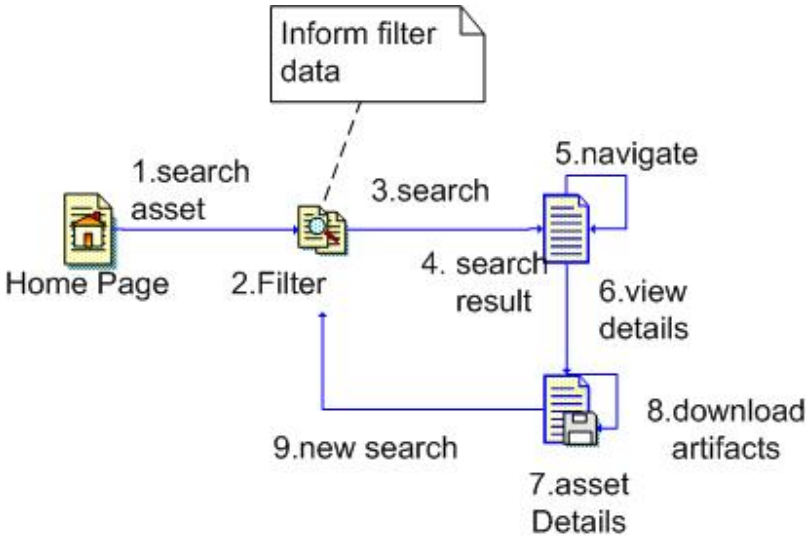


Figura 4.4 - Busca e recuperação: Fluxo navegacional

Os passos a serem executados são os seguintes:

1. Após ter logado no repositório, o usuário acessa a opção de busca, existente na tela principal;
2. O sistema apresenta uma tela com um filtro de busca a ser preenchido. Existem duas opções de busca de ativos no repositório implementado: (i) Busca por palavra chave e (ii) Busca avançada.
    i. Na busca por palavra chave, basta colocar uma ou mais palavras no campo de pesquisa, da mesma forma utilizada em um engenho de busca como o Google[23], por exemplo. As palavras informadas podem ser combinadas com uma série de operadores (AND, OR, NOT, etc);
    ii. Na busca avançada, um filtro de busca como o da Figura 4.5 é apresentado. Nele, além do campo de pesquisa (Figura 4.5-A), o usuário pode montar uma combinação de campos de classificação de

[23] http://www.google.com

ativos (Figura 4.5-B), com possibilidade de restringir o resultado aos ativos certificados (Figura 4.5-C).

Figura 4.5 - Exemplo de filtro de busca

3. O usuário então informa os dados do filtro de busca e executa a pesquisa;
4. O sistema exibe o resultado da busca, apresentando as versões dos ativos que satisfazem o filtro de busca especificado. Como pode ser visto na Figura 4.6, o resultado apresenta o ativo (Figura 4.6-A) e as versões de cada ativo que satisfazem o filtro de busca são agrupadas pelo nível de qualidade das mesmas (Figura 4.6-B), que será visto no próximo cenário de uso. Outras operações podem ser efetuadas pelo usuário nesta tela, tais como: Registrar erros (Figura 4.6-C), Registrar feedback de uso (Figura 4.6-D) e Registrar interesse no ativo para futuras notificações (Figura 4.6-E). O tipo do ativo e o repositório (catálogo) onde ele se encontra também são apresentados (Figura 4.6-F).

Figura 4.6 - Tela de resultado de busca

5. O usuário navega pelo resultado da busca e encontra o ativo de seu interesse;
6. Ele clica na versão "1.0" do "*File Transfer Library*" para visualizar os detalhes da mesma;
7. O sistema apresenta uma tela contendo os detalhes da versão selecionada. Parte das informações apresentadas pode ser vista na Figura 4.7 que mostra: (A) - uma descrição geral da versão, (B) - os classificadores que a caracterizam, (C) - os artefatos que compõem seu conteúdo, (D) - suas dependências internas e externas, (E) - seu nível de qualidade, (F) - as métricas de desenvolvimento cadastradas e, finalmente (G) - um lista de sugestões de outros ativos.
8. Por fim, o usuário efetua o *download* dos artefatos e retorna para a tela de busca;

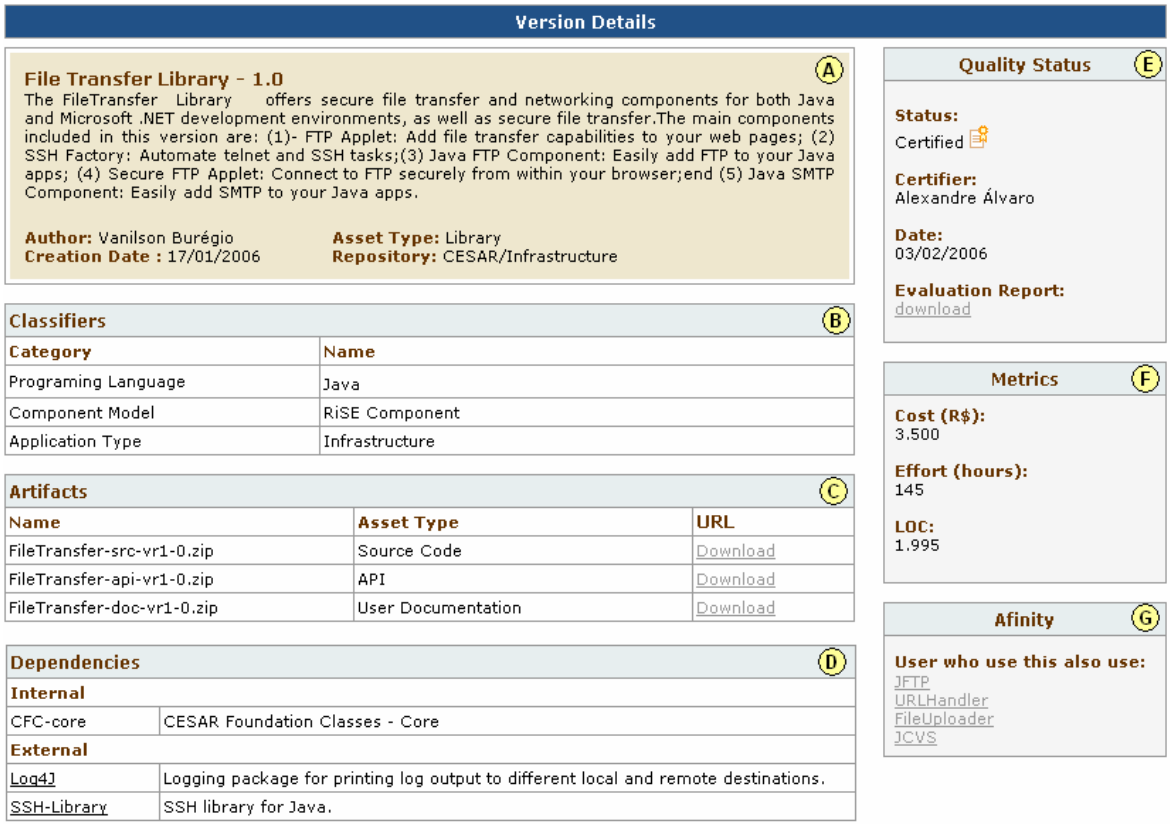


Figura 4.7 – Tela de visualização de detalhes de um ativo

4.1.3 Cenário 3: certificar versão de um ativo

Ao ser inserido no repositório, a versão de um ativo permanece em um estado de espera que indica que a mesma ainda não foi certificada. O usuário, com perfil de certificador, verifica nos repositórios que possui acesso o conjunto de ativos que estão aguardando avaliação. Ele julga que um dos ativos é potencialmente reusável e inicia o processo de avaliação (teste, inspeção de código, etc). Após a execução desse processo é gerado um relatório de avaliação e um novo status de certificação é atribuído ao ativo avaliado. O processo de certificação utilizado nesta implementação pode ser visto com mais detalhe no Apêndice C. Um resumo dos status de certificação que podem ser atribuídos aos ativos é apresentado a seguir.

Status de certificação

No processo adotado nesta implementação, as versões de um ativo podem assumir os seguintes status:

- **Aguardando avaliação** – ativo que ainda não foi avaliado ou recentemente submetido para uma re-avaliação;
- **Em avaliação –** ativo atualmente sendo avaliado;
- **Necessita melhorias –** ativo já avaliado e com deficiências constatadas;
- **Documentação insuficiente -** ativo recusado pelo processo de certificação, pois não possui a documentação necessária para execução do processo;
- **Certificado –** ativo avaliado e certificado pelo processo de certificação utilizado.

A seguinte máquina de estados pode ser observada no processo adotado:

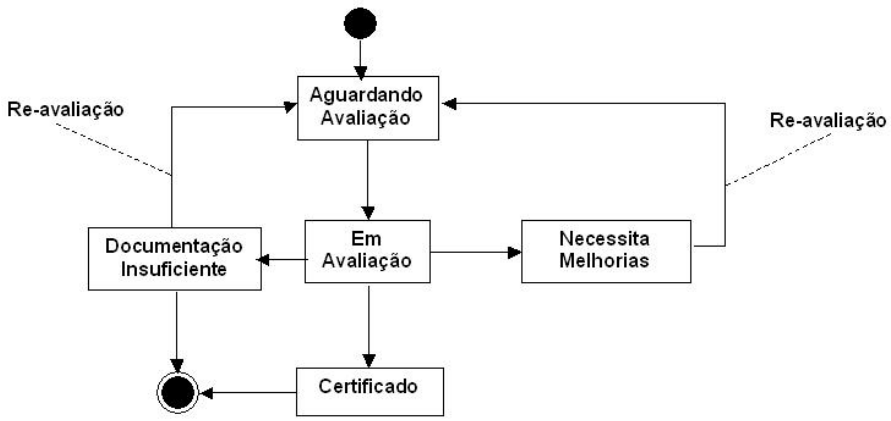


Figura 4.8 – Certificação: Máquina de estados

Duas considerações importantes devem ser ressaltadas quanto à certificação: (i) após a avaliação é gerado um relatório que é disponibilizado junto com a versão do ativo avaliada; (ii) tanto no status "*Necessita melhoria*" quanto no status "*Documentação insuficiente*" é possível que o produtor do ativo realize as modificações necessárias e submeta-o novamente para a avaliação.

Fluxo Navegacional

A Figura 4.9 apresenta o fluxo de navegação a ser executado por um usuário que deseja certificar um ativo do repositório que está “aguardando avaliação”.

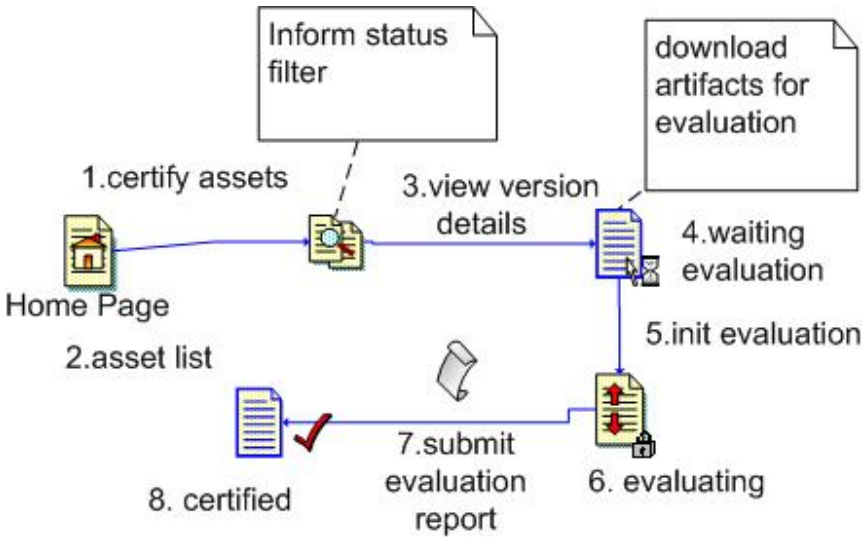


Figura 4.9 - Fluxo navegacional: certificar ativo

Os passos a serem executados são os seguintes:

1. Após ter logado no repositório, o usuário certificador acessa a opção de certificar ativo, presente na tela principal;
2. O sistema apresenta um filtro a ser preenchido para listagem dos ativos que podem ser certificados pelo usuário. A tela apresentada é similar a mostrada na Figura 4.10.
3. O usuário então seleciona a versão do ativo a ser certificada.
4. O sistema apresenta a tela de visualização de detalhes da versão selecionada que, além de outros metadados, mostra o status atual de certificação da versão (Figura 4.10 - A) e o histórico de alterações desse status (Figura 4.10 - B). No exemplo da Figura 4.10 o ativo foi re-submetido para uma nova avaliação, após o resultado da avaliação anterior (Figura 4.10- C) indicar a necessidade de melhorias.

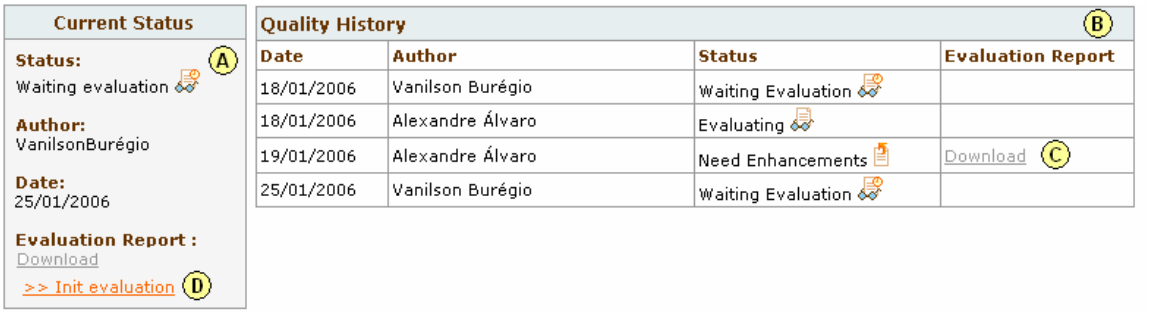


Figura 4.10 - Seções da tela de detalhes da certificação de um ativo

5. O usuário efetua o *download* dos artefatos da versão avaliada e acessa a opção de iniciar avaliação presente na tela de detalhes (Figura 4.10 - D). A partir desse momento, o ativo não mais poderá ser modificado por outro usuário e passa para o status "Em avaliação", de acordo com a máquina de estados já apresentada nesta Seção;
6. Durante a avaliação, um conjunto de características do ativo são analisadas e após esse processo um relatório de avaliação é gerado. O Apêndice C apresenta as características analisadas e os passos envolvidos no processo de avaliação;
7. Com o relatório de avaliação gerado, o certificador anexa-o ao ativo avaliado e informa o resultado da avaliação (Figura 4.11). O sistema então atualiza o status de certificação da versão do ativo e notifica o seu responsável e os usuários que registraram interesse no ativo em questão.

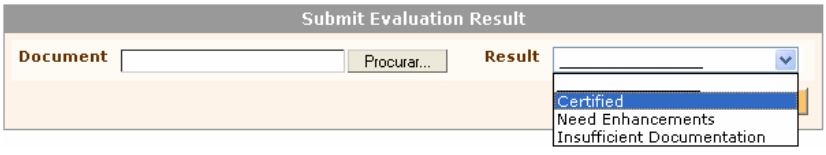


Figura 4.11 – Tela de anexação de relatório de certificação

Outro cenário de utilização freqüentemente executado no módulo de gerenciamento é a geração de relatórios. A execução deste cenário é simples, pois envolve basicamente 2 passos: (i) selecionar o tipo de relatório e (ii) informar os dados de filtragem do mesmo. Dada esta simplicidade, foi decidido não descrever

estes passos aqui e, ao invés disto, foi criado o Apêndice B que explica os principais tipos de relatórios considerados na solução e apresenta exemplos de saída destes. Este Apêndice serve de referência para quem deseja especificar os relatórios de um repositório de reuso como o proposto nesta dissertação.

As próximas Seções apresentam algumas avaliações que foram realizadas durante diferentes fases do desenvolvimento da solução proposta. Essas avaliações incluem: (1) avaliação da arquitetura, (2) avaliação da usabilidade. É importante ressaltar que o objetivo aqui é apresentar como estas avaliações contribuíram para a identificação de problemas na solução (antes e depois de sua implementação), bem como, reforçar a idéia que os métodos utilizados podem ser aplicados em outros contextos industriais. A próxima Seção considera a avaliação de arquitetura, abordando a sua importância e apresentando desde a análise e escolha do método de avaliação utilizado até os resultados obtidos com sua aplicação.

4.2 Avaliação de Arquitetura

A arquitetura de software está preocupada sobre como sistemas de software estão estruturados para melhor alcançar a qualidade do software [Bass et al., 1998], [Clements et al., 2002a]. Durante as últimas décadas, o projeto de arquitetura de software tem desempenhado um papel fundamental no desenvolvimento de sistemas. Isso é justificado pelo fato de que os efeitos da arquitetura em um sistema ou projeto são intensos, e arquiteturas inadequadas quase sempre causam catástrofes em projetos. Logo, a qualidade de um produto de software deve ser garantida desde cedo, durante o projeto da arquitetura inicial do sistema [Clements et al., 2002a].

Neste sentido, a disciplina de avaliação de arquitetura de software tem emergido como uma maneira de predizer a qualidade de um produto de software a partir da sua descrição arquitetural. De fato, alguns experimentos industriais mostraram que a aplicação de métodos de avaliação de arquitetura trás ganhos consideráveis durante o desenvolvimento de sistemas de software. Em [Maranzano et al., 2005], por exemplo, um grupo de pesquisadores da indústria relata suas

experiências em incorporar um processo de avaliação de arquitetura de software na metodologia de desenvolvimento de empresas como *AT&T*[24] e *Lucent Technologies*[25]. Neste trabalho, eles comprovam que o uso de tal abordagem nestas empresas gerou um ganho de cerca de 1 milhão de dólares em cada projeto que teve seus problemas identificados e resolvidos previamente. Os projetos considerados possuíam em média 100.000 linhas de código não comentadas.

Sendo assim, podemos concluir que tão importante quanto definir a arquitetura de um sistema de software é tentar realizar uma avaliação do que foi definido, afim de que as falhas encontradas sejam resolvidas antes que se tornem grandes problemas no futuro.

Sobre esta motivação, esta Seção apresenta um experimento industrial envolvendo a utilização de um processo de avaliação de arquitetura para avaliar a arquitetura do repositório proposto. É importante ressaltar que essa avaliação ocorreu logo após a definição inicial da arquitetura do repositório e acarretou em mudanças que já foram, em parte, incorporadas na arquitetura apresentada no Capítulo 4.

Metodologia

O processo de avaliação iniciou com uma mudança cultural focada em arquitetura de software. Inicialmente, o time envolvido na avaliação estudou alguns processos de desenvolvimento de arquitetura e realizou um *workshop*[26] sobre eles. Após isso, estudos envolvendo métodos de avaliação de arquitetura foram então realizados. Convencidos dos benefícios da avaliação, o time iniciou o processo escolhendo o método de avaliação apropriado.

Para que este processo possa ser replicado em outros contextos, esta Seção aborda desde os critérios utilizados para a escolha do método de avaliação até os resultados obtidos com a sua aplicação. Mais detalhes a respeito do experimento podem ser encontrados em [Burégio, 2006]. A próxima Seção apresenta uma visão

[24] http://www.att.com/
[25] http://www.lucent.com/
[26] http://www.rise.com.br/seminars/workshop.pdf

geral sobre os métodos de avaliação de arquitetura e descreve brevemente os três métodos que foram analisados antes da realização do experimento.

4.2.1 Métodos de avaliação

O processo de avaliação de arquitetura de software tipicamente inclui a aplicação de um método de avaliação. Existem diversos métodos disponíveis com diferentes técnicas e objetivos [Kazman et al., 1994] [Bengtsson & Bosch, 1998] [Kazman et al., 1999] [Clements et al., 2002a].

Em geral, podemos dividir as técnicas usadas pelos métodos de avaliação existentes em dois tipos [Clements et al., 2002a]: (1) *técnicas de medição* e (2) *técnicas baseadas em questionamentos.*

Técnicas de medição envolvem a aplicação de algum mecanismo de medição de artefatos de software com a finalidade de avaliar *atributos de qualidade* [Bass et al., 1998] específicos. Isto inclui mecanismos para extração de métricas, simuladores, protótipos e experimentos com o sistema que se deseja avaliar. Nota-se que estas técnicas são aplicáveis quando tem-se algo implementado, ou seja, pelo menos parte da arquitetura deve ter sido construída. Um exemplo comum é avaliar a *performance* de uma aplicação já implementada.

Por outro lado, as *técnicas baseadas em questionamentos* são mais subjetivas do que as *técnicas de medição* e envolvem o uso de questionários, *checklist* e cenários para investigar como a arquitetura satisfaz os requisitos desejados. Logo, essas técnicas podem ser aplicadas sem a necessidade de termos uma implementação concreta da arquitetura e são realizadas antes da implementação da aplicação.

Embora os métodos de avaliação de arquitetura possam variar quanto às técnicas que utilizam, pode-se dizer que, de uma maneira geral, os métodos possuem um propósito em comum [Kazman et al., 2005]. Este propósito é: investigar se uma determinada *arquitetura* satisfaz os *objetivos de negócio* de um *sistema* para o qual foi projetada. Desta forma, a fim de alcançar este propósito, o método de avaliação é guiado por algumas *propriedades*. Estas propriedades

definem o escopo da avaliação, pois determinam os critérios sob os quais o sistema será avaliado. A Figura 4.12 resume a idéia básica de qualquer método de avaliação.

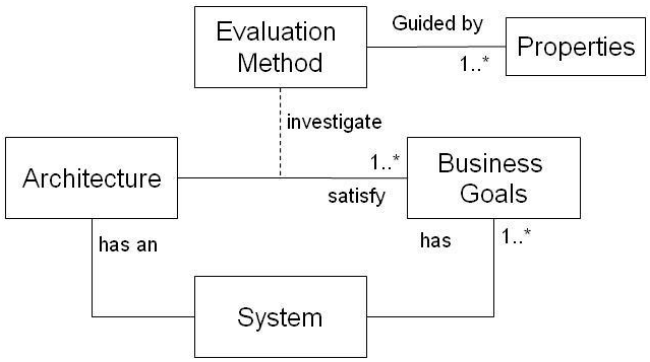


Figura 4.12 - Método genérico de avaliação de arquitetura

No caso do experimento a ser descrito, foram levados em consideração métodos baseados em cenários, já que no período de sua aplicação, o objetivo era o de avaliar a arquitetura de um sistema ainda não implementado.

Um cenário, neste contexto, ilustra os tipos de atividades que um sistema deve suportar. Eles também ilustram os tipos de mudanças que os clientes antecipadamente confirmam que serão feitas no sistema, por isso, também são conhecidos como cenários evolutivos ou cenários de mudanças. Um exemplo de cenário pode ser: "*Qual o impacto na arquitetura caso ocorra uma mudança no modelo de troca de mensagens entre os componentes*".

Vale ressaltar que o desenvolvimento desses cenários é crucial para capturar os principais usos do sistema, seus usuários, antecipar mudanças no sistema, e qualidades que o sistema deve satisfazer agora e no futuro. Estes cenários devem ser elicitados por todos os *stakeholders* do sistema, logo, representam questões sob diferentes pontos de vista e, na maioria das vezes, trazem informações bem específicas do sistema avaliado. Um exemplo de um cenário elicitado para a arquitetura do repositório apresentada no Capítulo 3 poderia ser: "*Permitir que o módulo AssetSearcher suporte a busca em código compilado usando um mecanismo de reflection*".

A seguir é apresentado uma breve descrição de alguns métodos baseados em cenários e o processo de escolha utilizado para selecionar o método utilizado na avaliação de arquitetura.

Métodos analisados

Em particular, foram analisados os três métodos usados nos estudos práticos descritos em [Clements et al., 2002a]: (1) *SAAM* [Kazman et al., 1994], (2) *ATAM* [Kazman et al., 1999] [Kazman et al., 2000], e (3) *ARID* [Clements et al.,2002b]. A seguir uma breve descrição de cada um desses métodos é apresentada:

1. **SAAM** (*Software Architecture Analysis Method*): O *SAAM* é um método baseado em cenários cujo principal foco é analisar uma arquitetura a respeito dos requisitos de modificabilidade e funcionalidade. Porém, ele também pode ser aplicado para analisar outros atributos de qualidade, tais como portabilidade, extensibilidade e integrabilidade;
2. **ATAM** (*Architecture Tradeoff Analysis Method*): Similar ao SAAM, o ATAM também é um método baseado em cenários, porém, é mais complexo se comparado ao SAAM. A aplicação do ATAM revela quão bem uma arquitetura satisfaz os requisitos de qualidade desejados, além de identificar os riscos arquiteturais e os conflitos (*trade-offs*) existentes para se alcançar tais requisitos;
3. **ARID** (*Active Reviews for Intermediate Designs*): Este método apresenta uma abordagem mais leve para avaliar um projeto parcial de uma arquitetura ainda no início de sua especificação. O ARID foca na avaliação da adaptabilidade da arquitetura frente aos objetivos de negócio.

4.2.2 Escolha do Método a ser utilizado

Após um estudo sobre os três métodos apresentados na Seção anterior, foi preciso decidir que método deveria ser utilizado na avaliação. Na literatura, existem dois importantes *surveys* sobre métodos de avaliação de arquitetura [Dobrica & Niemela, 2002][Barbar et al., 2004], que ajudaram nessa decisão. Além disso,

Kazman et al. em [Kazman et al., 2005] apresenta um resumo que define os critérios fundamentais a serem utilizados na comparação de métodos de avaliação de arquitetura.

Baseado nestes trabalhos, foram priorizados 6 (seis) requisitos que ajudaram a definir o método de avaliação a ser utilizado: (1) Maturidade do método, (2) Objetivo do método, (3) Atributos de qualidade considerados, (4) Estágio de aplicabilidade, (5) Maturidade da arquitetura, (6) Recursos necessários. Segue uma breve definição desses requisitos:

1. **Maturidade do método**: este requisito tem por objetivo analisar se o método já foi validado na prática (em casos industriais);
2. **Objetivo do método**: analisa os objetivos específicos do método;
3. **Atributos de qualidade**: identifica que atributos de qualidade são levados em consideração pelo método;
4. **Estágio de aplicabilidade**: identifica quando (estágio do projeto) o método deve ser aplicado;
5. **Maturidade da arquitetura:** analisa qual é a maturidade (fase de desenvolvimento) da documentação da arquitetura de software da solução; e
6. **Recursos necessários**: analisa os recursos necessários para a realização da avaliação, tais como, a quantidade de *homems-dia* necessária e o tamanho da equipe de avaliação.

A Tabela 4.1 apresenta os métodos comparados e suas relações com os requisitos listados.

Tabela 4.1 - Comparação dos métodos de avaliação

Requisitos	SAAM	ATAM	ARID
Maturidade do método	Validado em vários domínios	Continuamente em validação	Ainda não validado extensivamente
Objetivo do Método	Identificação de riscos e análise de adequação da arquitetura	Foco em sensibilidade da arquitetura e análise de *Trade-off*	Valida a viabilidade do projeto arquitetural
Atributos de qualidade	Foco em modificabilidade e funcinalidade	Multiplos atributos	Adaptabilidade do projeto
Estágio de aplicabilidade	A partir do momento que as funcinonalidades forem atribuidas aos módulos	Após o projeto detalhado da arquitetura	Durante o projeto dos componentes
Maturidade da arquitetura	Requer visão lógica e visão de módulos	Visões de processos, fluxo de dados, use-case, física e de modulos	Projeto preliminar
Recursos necessários*	Pelo menos: 2 dias. 3 pessoas no time de avaliação + *stakeholders*	Pelo menos: 3 dias. 4 pessoas no time de avaliação + *stakeholders*	Pelo menos: 2 dias. 2 pessoas no time de avaliação + *stakeholders*

* Não considerando tempo de preparação (fase de estudo do método)

Antes de realizar a análise comparativa apresentada na Tabela 4.1, a primeira intenção era usar o ARID, isso porque ele é um método mais leve e teoricamente exigiria menos tempo de preparação. Porém, depois da análise comparativa, foi decidido usar o SAAM no experimento, pois, como pode ser verificado na próxima Seção, a maior intenção da avaliação foi analisar os requisitos de modificabilidade e funcionalidade, e tais requisitos não são considerados pelo ARID. Por outro lado, a avaliação de arquitetura utilizando esta abordagem era uma experiência pioneira no contexto do projeto. Sendo assim, um experimento envolvendo um método mais complexo como o ATAM poderia gerar dificuldades naquele primeiro momento.

De fato, Clements et al. [Clements et al., 2002a] sugerem o uso do SAAM em uma experiência inicial e complementam dizendo (pp.212):

"O SAAM é um método simples, fácil de entender e fácil de ser executado com uma quantidade relativamente pequena de treino e preparação. Ele é uma boa opção

para iniciar, caso você nunca tenha feito uma avaliação de arquitetura antes, particularmente se você estiver interessado em avaliar modificabilidade e funcionalidade."

Após a escolha do método de avaliação apropriado, o time de avaliação foi então definido. Este time foi composto por desenvolvedores e arquitetos provenientes de outras equipes de desenvolvimento do C.E.S.A.R. Isso teve por objetivo manter o processo de avaliação neutro, ou seja, sem influências da equipe que projetou a arquitetura. A Seção a seguir apresenta o processo de avaliação adotado.

4.2.3 Processo de avaliação

O processo de avaliação foi dividido em 3 reuniões e envolveu os passos apresentados na Figura 4.13.

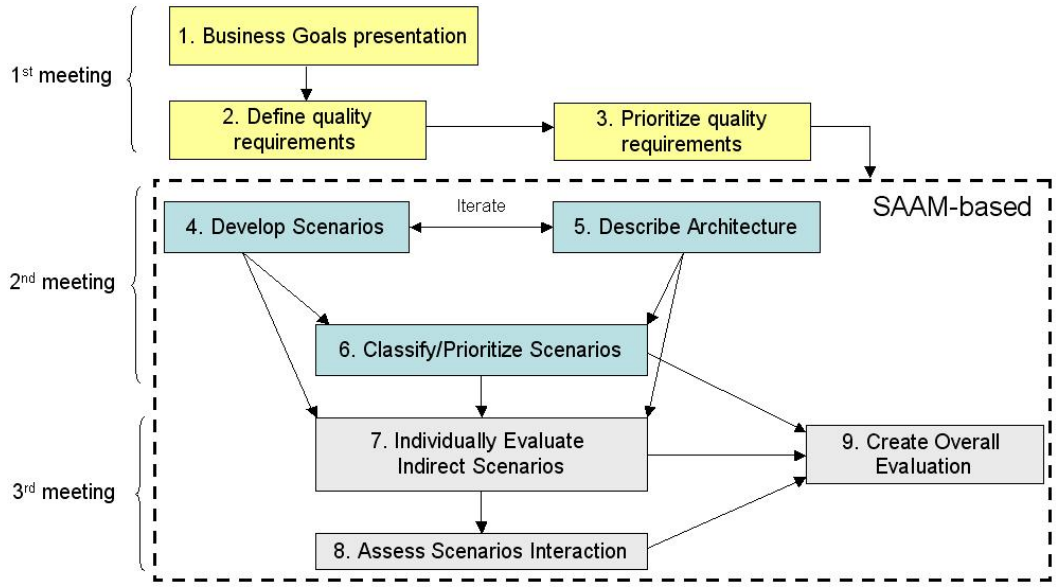


Figura 4.13 - Processo de avaliação de arquitetura

Primeira reunião

Na primeira reunião, houve a participação de 10 (dez) *stakeholders* envolvidos com o sistema: 5 (cinco) com uma visão técnica e 5 (cinco) com interesses de gerenciamento e negócios. Nesta ocasião, o foco foi formalizar os objetivos do

negócio a serem utilizados no processo de avaliação e os atributos de qualidade que seriam avaliados. As etapas dessa reunião foram as seguintes:

1. Apresentação dos objetivos de negócio (Duração: 30 min). Este foi o primeiro passo da reunião e teve a intenção de definir e apresentar os objetivos de negócio do repositório a ser construído. Este passo serviu para que os participantes da avaliação entendessem o contexto do sistema, suas principais funcionalidades, seus *stakeholders*, e também as limitações técnicas, econômicas e de cronograma existentes para a execução do projeto. Todos esses aspectos foram apresentados para ajudar na definição dos atributos de qualidade a serem considerados no processo de avaliação. Além disso, foi feita uma breve explicação das atividades a serem executadas durante a avaliação;

2. Definição dos atributos de qualidade (Duração: 20 min). Nesta etapa, a equipe definiu a seguinte lista de atributos de qualidade que poderiam ser avaliados: disponibilidade, funcionalidade, modificabilidade, manutenabilidade, desempenho, escalabilidade, segurança, testabilidade, e usabilidade. Estes atributos podem ser vistos como requisitos arquiteturais que correspondem aos critérios sob as quais a arquitetura será avaliada;

3. Priorização dos atributos de qualidade. Com o objetivo de definir a prioridade dos atributos de qualidade listadas no passo anterior, foi iniciado um processo de votação. Neste processo, cada stakeholder tinha direito a 3 (três) votos e deveria distribuí-los entre os atributos de qualidade de seu interesse. Este processo utilizou um processo de votação individual, para evitar influência na votação dos outros participantes. A Tabela 4.4.2 apresenta o resultado.

Tabela 4.4.2 – Resultado da priorização dos atributos de qualidade

Classificação	Atributo	# votos
1º	Modificabilidade	7
2º	Funcionalidade	6
	Usabilidade	6
3º	Performance	3
	Disponibilidade	3
4º	Escalabilidade	2
	Testabilidade	2
5º	Segurança	1
6º	Manutenabilidade	0

As definições consideradas para os atributos foram:

- **Modificabilidade**: atributos do software que evidenciam o esforço necessário para modificá-lo, remover seus defeitos ou adaptá-lo a mudanças;
- **Funcionalidade**: conjunto de atributos que evidenciam a existência de um conjunto de funções e suas propriedades especificadas. As funções são as que satisfazem as necessidades explícitas ou implícitas;
- **Usabilidade**: conjunto de atributos que evidenciam o esforço necessário para se poder utilizar o software, bem como o julgamento individual desse uso, por um conjunto explícito ou implícito de usuários;
- **Desempenho:** conjunto de atributos que interferem diretamente nos tempos de resposta de um sistema;
- **Disponibilidade:** refere-se ao conjunto de atributos que interferem na probabilidade de que um sistema esteja funcionando e pronto para uso em um dado instante de tempo;
- **Escalabilidade:** capacidade que um sistema possui de se expandir, de forma a permitir o atendimento das necessidades da empresa pelo crescimento do número de usuários do sistema, ou também pelo aumento das informações a serem processadas;
- **Testabilidade**: atributos do software que evidenciam o esforço necessário para validar o software modificado;
- **Segurança:** refere-se ao conjunto de propriedades da arquitetura que interferem diretamente na garantia da integridade dos dados da aplicação e no acesso aos mesmos;
- **Manutenabilidade:** refere-se ao grau de facilidade de um sistema em ser mantido, modificado ou adaptado. É um conceito muito próximo ao de modificabilidade;

Segunda reunião

Esta foi a primeira reunião técnica do processo de avaliação. A agenda foi baseada na proposta do método SAAM, porém, houve algumas adaptações para a realidade do projeto. Em cerca de 20 (vinte) minutos, o líder da avaliação apresentou os

objetivos da avaliação e forneceu uma breve descrição do processo a ser utilizado. Depois disso, o arquiteto do sistema teve 10 (dez) minutos para fornecer uma visão geral das funcionalidades do sistema a ser avaliado. A idéia era desenvolver alguns cenários, por enquanto, sem nenhuma influência da arquitetura existente.

4. Desenvolvimento de cenários (1ª iteração). Os cenários do repositório foram desenvolvidos em 2 iterações. Nesta primeira iteração, os cenários elicitados foram baseados apenas nas funcionalidades do sistema e no conhecimento geral a respeito do domínio da aplicação, que foram apresentados inicialmente. Esta iteração durou cerca de 1 hora e funcionou como uma espécie de *brainstorm*, sem muitas críticas ou restrições a respeito dos cenários elicitados.

5. Detalhamento da arquitetura. Neste passo, uma visão geral da arquitetura foi descrita pelo arquiteto através de uma visão de módulos. Esta etapa durou cerca de 30 minutos e teve por objetivo apresentar os principais módulos do repositório. Como se trata de um processo iterativo, a descrição da arquitetura apresentada levou os *stakeholders* a pensar sobre novos cenários voltados para as características específicas da arquitetura levada em consideração. Com a primeira definição de cenários (antes da apresentação dos detalhes da arquitetura), o time elicitou cerca de 20 cenários, e após a apresentação dos detalhes da arquitetura, foram elicitados mais de 50;

6. Classificação/Priorização dos cenários. Após a elicitação dos cenários, o passo seguinte foi classificá-los e depois priorizá-los. A classificação consistiu em dividir os cenários em duas categorias: (1) *cenários diretos* – representavam os cenários já contemplados pela arquitetura, e (2) *cenários indiretos* – representavam os cenários que não eram satisfeitos pela arquitetura proposta. Durante o processo de classificação, alguns cenários foram agrupados. Esse agrupamento teve por objetivo identificar cenários relacionados e em seguida generalizá-los em um cenário único. Só após isso foi iniciada a priorização dos cenários.

A priorização dos cenários focou nos cenários indiretos e permitiu definir quais os cenários mais *importantes* a serem tratados dentro do limite de tempo disponível para a avaliação. *Importante* neste caso foi definido inteiramente pelos

stakeholders e seus interesses. Seguindo sugestão apresentada por Clements et al. [Clements et al., 2002] (pp. 217), foi realizada uma votação individual onde cada participante teve direito a um número de votos que correspondeu a 30% do total de cenários indiretos elicitados. Cada pessoa pôde distribuir os votos de acordo com seus interesses, respeitando o limite de não atribuir mais de 1/3 de seus votos em um único cenário.

Terceira reunião

7. Avaliação individual dos cenários priorizados - Uma vez que o conjunto de cenários indiretos foi definido, esses cenários foram mapeados na descrição arquitetural apresentada. No caso dos cenários diretos, o arquiteto demonstrou como os cenários seriam executados pela arquitetura proposta. No caso dos cenários indiretos, o arquiteto descreveu como a arquitetura precisaria ser modificada para suportar tais cenários.

Este passo foi de fundamental importância para que os revisores e os *stakeholders* tivessem um melhor entendimento da estrutura da arquitetura do repositório e da iteração dinâmica de seus componentes.

O processo de mapeamento dos cenários na arquitetura explicitou os pontos fracos do projeto arquitetural e também de sua documentação, além de identificar novos requisitos.

Para cada cenário indireto, foram listadas as mudanças arquiteturais necessárias para suportar o cenário. Ao final desta etapa, foi gerada uma planilha contendo a descrição dos cenários e o conjunto de mudanças necessárias.

8. Avaliar interações entre cenários. Quando 2 ou mais cenários indiretos requerem mudanças em um único componente de uma arquitetura, é dito que esses cenários interagem neste componente. Nesta etapa, foram identificadas as interações entre os cenários já mapeados na arquitetura.

A identificação destas interações entre cenários foi importante porque reforçou como as funcionalidades do repositório estavam alocadas no projeto arquitetural. Além disso, a interação entre cenários semanticamente não relacionados mostrou que componentes da arquitetura estavam tratando

funcionalidades semanticamente distintas. Ou seja, áreas de interação muito intensa de cenários revelaram uma separação potencialmente pobre de conceitos em um componente. O exame dessas interações entre cenários também revelou que a arquitetura não estava documentada no nível correto de decomposição estrutural.

9. Criação do resultado geral da avaliação. Este passo consistiu basicamente da documentação do processo de avaliação realizado. Para isso foi gerado um relatório que apresentou o resultado da avaliação, fornecendo uma visão geral dos principais problemas encontrados na arquitetura e especificando as mudanças estruturais a serem realizadas na mesma.

A Seção a seguir resume os resultados obtido com o processo de avaliação de arquitetura realizado.

4.2.4 Resultados da avaliação de arquitetura

A avaliação efetuada na arquitetura do repositório revelou alguns problemas na documentação, clareza dos seus requisitos e algumas questões técnicas de implementação.

Documentação

Foi constatado que a documentação da arquitetura do sistema deveria ser mais detalhada. A documentação utilizada apresentava a arquitetura com a visão de módulos e não era possível identificar a comunicação existente entre os componentes da arquitetura. A falta de documentação de alguns pontos gerou dúvidas no momento da avaliação, pois não era possível identificar como as funcionalidades estavam alocadas nos módulos do sistema, quais eram as dependências funcionais existentes entre os módulos especificados e nem como eles estavam organizados internamente. Em parte, esta dificuldade de documentação de arquitetura se deve ao fato desta área está começando a amadurecer. Atualmente, um bom guia prático, recomendado pelo *SEI*[27], para

[27] http://www.sei.cmu.edu/

representação e documentação de arquitetura de software pode ser encontrado em [Clements, 2002b].

Requisitos

Um dos benefícios de se realizar qualquer avaliação de arquitetura é a melhoria de comunicação entre os *stakeholders*, resultando num melhor entendimento dos requisitos. Durante a avaliação foi percebido que alguns requisitos do repositório, que possuem impacto em sua arquitetura, não foram especificados com precisão ou ainda não foram definidos. Por exemplo, não foi especificado nenhum requisito de tempo para o sistema, tais como tempo de resposta máximo que uma determinada operação do sistema deve ser realizada. Adicionalmente, não havia nenhum requisito que estabelecesse o nível de disponibilidade que o sistema deveria possuir.

Mudanças estruturais

O conjunto de problemas relacionados apresentados anteriormente não é o único resultado importante da avaliação de arquitetura. Algumas mudanças estruturais importantes no projeto da arquitetura foram levantadas a partir das discussões que ocorreram durante o processo de mapeamento dos cenários na arquitetura inicialmente proposta. Dentre os resultados mais importantes, pode-se citar: a criação dos *listeners* previstos em alguns módulos da arquitetura; o componente para extração automática de ativos; um componente para rankeamento dos resultados da busca baseado no feedback fornecido pelos usuários, entre outros.

4.3 Decisões Técnicas

Durante o desenvolvimento do repositório, foram levantadas algumas questões que precisaram de uma análise mais detalhada. Esta Seção apresenta as principais questões levantadas durante o desenvolvimento e implantação do produto. Tal análise tem por objetivo esclarecer algumas decisões tomadas e servir como guia para a construção de repositórios deste tipo. As questões estão agrupadas a seguir:

1. Modelo de ativos

Uma das primeiras questões tratadas na implementação de um repositório de reuso é decidir como será a estrutura de um ativo armazenado no mesmo [Ezran et al., 2002]. Para isso, foi feito um estudo de como componentes de software são empacotados e caracterizados em algumas abordagens e soluções existentes.

Dentre as abordagens analisadas, a que se mostrou mais promissora foi a adotada pelo *RAS* (*Reusable Asset Specification*)[28]. O RAS é uma especificação de empacotamento de ativos de software da OMG que foi proposta por um consórcio formado por empresas de influência no mercado, como *IBM*[29] e *Borland*[30]. Ele utiliza documentos XML para a representação dos metadados de um ativo. As informações mínimas que um ativo deve conter estão definidas no *Core RAS* que pode ser estendido para criar outros modelos de componentes.

Dessa forma, o modelo de ativo, adotado no repositório implementado, tentou preservar o conteúdo de informações requerido pelo *Core RAS*, apesar de ter modificado sua estrutura e acrescentado novos atributos. Esta compatibilidade com o *Core RAS* é de suma importância para permitir a futura interoperabilidade entre o repositório e outras ferramentas do mercado. Neste sentido, um mecanismo de exportação/importação de dados está sendo criado.

A Figura 4.14 identifica as principais seções e os elementos tratados em um ativo armazenado no repositório. A mudança estrutural mais importante foi considerar que um ativo é formado por uma coleção de versões. Tanto o ativo, quanto suas versões possuem atributos gerais associados (descrição, autor, tipo, etc). As quatro principais seções de uma versão de um ativo incluem:

- **Classificação** – lista um conjunto de descritores para a classificação de um ativo;
- **Conteúdo** – descreve os artefatos do ativo;
- **Versões relacionadas** – descreve as relações entre a versão com outras versões; e
- **Informações extras** – possui um conjunto de informações úteis como métricas, feedback do usuário, bugs e informações de certificação da versão.

[28] http://www.omg.org/technology/documents/formal/ras.htm
[29] http://www.ibm.com
[30] http://www.borland.com

O modelo de classes detalhado contendo as entidades mais importantes do repositório pode ser visto no Apêndice A.

Figura 4.14 – Estrutura de ativo adotada

2. Controle de acesso aos ativos

Outro problema que deve ser levado em consideração na construção de um repositório de reuso é a relação entre as regras de acesso aos ativos e a estrutura da empresa onde o repositório será implantado.

Na experiência deste trabalho, a empresa era uma fábrica de software que possuía vários times de desenvolvimento atuando, algumas vezes, em projetos destinados a clientes concorrentes. Além disso, havia a necessidade de interagir com times de desenvolvimento terceirizados (*outsourcing*), que não deveriam ter acesso a alguns ativos privados. Neste tipo de organização, é preciso que o repositório seja capaz de armazenar informações a respeito dos grupos organizacionais existentes e, a partir daí, conseguir definir o escopo de acesso desses grupos.

Na implementação realizada, os ativos estão agrupados em *repositórios virtuais* (catálogos) que possuem 2 tipos de visibilidade: pública e privada. Catálogos públicos representam repositórios visíveis a todos os usuários do sistema. Por outro lado, um catálogo privado só pode ser acessado pelos usuários que pertencem aos grupos organizacionais associados a ele. Nessas associações, o tipo de permissão (leitura/escrita) de cada grupo é estabelecido pelo *Catalogue manager* do módulo de gerenciamento, explicado na Seção 3.2.5.

3. Sistemas legados

O processo de implantação de aplicações corporativas, como o repositório proposto, geralmente precisa levar em consideração os sistemas legados e políticas existentes no contexto organizacional onde serão implantados. Logo, é de fundamental importância analisar as necessidades existentes na organização e propor mecanismos que possam permitir a fácil integração do repositório ao seu contexto, de modo a torná-lo uma ferramenta menos intrusiva.

Neste sentido, uma necessidade tratada no repositório implementado foi evitar o cadastro manual dos usuários. Para isso foi disponibilizada uma interface para um adaptador que representa uma abstração da base legada de usuários da

empresa. O adaptador permite autenticar os usuários sem a necessidade de o mesmo ter sido cadastrado manualmente no sistema.

4. Interesses de usuários e envio de notificações

Como já mencionado no Capítulo anterior, deve ser possível aos usuários cadastrarem interesses e, em seguida, receberem notificações do repositório baseadas nesses interesses. Porém, ao definir os tipos de interesses que podem ser informados deve-se ter o cuidado de avaliar a qualidade das notificações geradas. O objetivo disso é evitar que o sistema envie muitas notificações que não são relevantes. Um exemplo disso foi notado quando foi permitido aos usuários registrarem interesses muito genéricos que eram satisfeitos pela maioria dos ativos do repositório. Neste caso, um número muito grande de notificações pouco relevantes foi gerado.

Um exemplo de notificação relevante é permitir que o usuário registre interesse de notificação quando ativos baixados por ele forem alterados. Outro exemplo é permitir o registro de interesse em um filtro de busca que não retornou resultado. Dessa forma, o usuário será notificado quando ativos que “casarem” com o filtro especificado forem inseridos no repositório.

4.4 Resumo do capítulo

Este Capítulo apresentou alguns exemplos de uso da ferramenta desenvolvida, discutiu o processo de avaliação de arquitetura realizado durante o desenvolvimento da solução e apresentou algumas decisões técnicas tomadas durante o desenvolvimento do repositório proposto. Todas as questões tratadas neste Capítulo servem de base para quem deseja construir ou adotar um repositório de reuso que suporte um processo de reutilização de software em um contexto corporativo.

O próximo Capítulo apresenta as conclusões finais deste trabalho, suas principais contribuições e direções para trabalhos futuros.

5 Conclusões e Trabalhos Futuros

Uma vez que o repositório proposto foi especificado, projetado e implementado, algumas conclusões e direções para trabalhos futuros podem ser apontadas.

Desta forma, este Capítulo apresenta as conclusões finais deste trabalho e está organizado da seguinte maneira: a **Seção 5.1** apresenta um resumo das principais conclusões deste trabalho. A **Seção 5.2** trata alguns trabalhos relacionados. A **Seção 5.3** aponta um conjunto de trabalhos futuros, e finalmente, a **Seção 5.4** contem a conclusão final da dissertação.

5.1 Principais conclusões

Esta Seção resume as principais conclusões deste trabalho. Estas conclusões estão enumeradas abaixo:

- Repositórios representam um recurso precioso para facilitar o problema de localizar software reutilizável e formam a infra-estrutura básica de suporte ao processo de reutilização de software, porém o uso de repositórios para suportar um processo de reuso tem muitas vezes falhado (Seção 1.1);
- Na literatura, o foco dos trabalhos envolvendo repositórios de reuso está, quase sempre, voltado a questões de busca e recuperação de componentes e, na maioria das vezes, os aspectos organizacionais e de qualidade não são explorados adequadamente (Seção 2.2);
- Na prática, a maioria das alternativas não suporta eficientemente os papéis desejáveis para um repositório de reuso e diversas questões, levantadas por

empresas que desejam construir ou adotar um repositório, continuam mal respondidas (Seção 2.3);

- O repositório proposto por esta dissertação possui uma arquitetura flexível que serviu de base para a implementação de um produto comercial. Este produto teve como objetivo preencher as principais lacunas identificadas durante a pesquisa das soluções existentes. Sua eficácia está sendo comprovada em um contexto real de uma fábrica de software com mais de 1500 desenvolvedores (Capítulo 3); e
- O processo de avaliação de arquitetura realizado no desenvolvimento do repositório proposto mostrou-se bastante útil. Ele confirmou os principais benefícios citados na teoria, tais como proporcionar um melhor entendimento da arquitetura a ser adotada, possibilitar melhorias na documentação e identificar os principais riscos arquiteturais (Seção 4.2).

5.2 Trabalhos Relacionados

Blaha et al. [Blaha et al., 1998], listam um conjunto de requisitos para um sistema de repositório de reuso. Contudo, os requisitos apresentados são totalmente focados na aplicação de repositório para apoiar um processo de engenharia reversa. Como resultado, as funções listadas são baseadas em teoria da transformação e incluem operações como mapeamento entre modelos e esquemas físicos.

Seacord [Seacord, 1999] verifica a viabilidade de utilização de tecnologias, como *traders*, *brokers*, serviços de localização e mecanismos de buscas, aliados a introspecção de código, a fim de criar um repositório de componentes distribuído e acessível através da Internet. Esse repositório recebeu a denominação de *Agora*. Mesmo sendo considerado uma evolução dos repositórios específicos, adotando os princípios de repositórios de referências, o sistema *Agora* apresentou muitas limitações. Por basear-se num mecanismo de busca utilizando introspecção computacional, o sistema apresentou limitações tanto semanticamente quanto na abrangência, uma vez que são recuperados apenas componentes binários.

Na Internet, existem alguns sistemas de repositórios disponíveis, como, por exemplo, o Jumbo![31] [Jumbo!] e o ZDNet[32]. Entretanto, esses sistemas estão mais direcionados para usuários de software que desejam obter ferramentas ou sistemas já prontos, não sendo útil para desenvolvedores. O repositório de software SourceForger[33] possui uma grande diversidade de projetos disponíveis através do modelo *open-source* [Raymond, 2001]. Nesse repositório, um sistema de busca simples é utilizado para consulta de projetos de desenvolvimento, porém, existe pouco suporte para explorar componentes de software, código fonte ou algoritmos reutilizáveis.

Alguns produtos voltados para o gerenciamento de componentes, também podem ser encontrados. O *Select component manager* (SCM) é um exemplo destes. Ele consiste de uma ferramenta que faz parte de um conjunto de soluções - *Select Component Factory* – que forma um ambiente de desenvolvimento de aplicações com suporte a reuso. O SCM possui funcionalidades que permitem a publicação, gerenciamento, busca, browsing e reuso de componentes. Esta ferramenta representa um gerenciador de componentes bastante completo, porém seu principal problema é o custo de sua licença e a limitação de sua compatibilidade, pois é uma aplicação *desktop* para Windows. Além disso, não possui suporte a um processo de certificação de componentes que garanta a qualidade dos componentes disponibilizados.

Atualmente, um conjunto de empresas está definindo uma especificação de acesso a repositórios de conteúdo. Essa iniciativa proposta ao JCP (Java Community Processs) está sendo coordenada por empresas como *Apache Software Foundation, Hewlett-Packard, IBM* e é denominada *Content Repository for Java Technology API*[34] O objetivo dessa API é possuir uma forma padronizada de acesso a esses repositórios que resulte em um impacto menor na integração entre aplicações produtoras de conteúdo e aplicações consumidoras. O principal problema dessa proposta é que a mesma não está ligada diretamente à especificação e documentação de componentes, e sim à manipulação de

[31] : http://www.jumbo.com
[32] http://www.zdnet.com
[33] http://sourceforge.net
[34] http://jcp.org/en/jsr/detail?id=170

repositório, logo, é preciso um esforço adicional para combinar sua especificação, com um modelo de ativos (como o RAS) para que se obtenha uma solução específica para reuso de software.

Analisando os principais trabalhos relacionados pode-se observar que não existe uma solução completamente equivalente à proposta por esta dissertação. Os resultados desta dissertação estão baseados em um trabalho de pesquisa que analisou o estado da arte e da prática no tocante ao uso de repositórios para suportar um processo de reutilização de software. Estes resultados envolvem, entre outras coisas: (i) a criação de uma taxonomia das soluções existentes, (ii) a definição dos papéis de um repositório de reuso, (iii) a especificação dos seus principais requisitos, (iv) o projeto de sua arquitetura, (v) a utilização - na indústria - de um processo de avaliação de arquitetura, (vi) a definição inicial de um processo de certificação de componentes, e (vii) a implementação de um produto comercial que pudesse comprovar, na prática, os benefícios do repositório proposto.

5.3 Trabalhos Futuros

A versão inicial do repositório implementado não cobre algumas áreas importantes que não foram explorados neste trabalho:

- **Distribuição do índice**: o requisito de escalabilidade, que é importante para o uso do repositório em diferentes contextos, só será satisfeito quando o índice de busca for distribuído entre servidores. O engenho de busca *Lucene*, adotado na implementação do repositório, provê maneiras de implementar esta funcionalidade. Assim, alguns ajustes devem ser feitos nos componentes *AssetSearcher* e *AssetIndex;*
- **Extração automática de métricas**: uma funcionalidade complementar é suportar a extração automática de métricas dos ativos inseridos no repositório. Dessa forma, quando um artefato de código, por exemplo, for inserido, o sistema automaticamente pode verificar métricas como LOC, complexidade ciclomática, acoplamento, etc;

- **Desenvolvimento de plug-ins:** para permitir a integração do repositório com diferentes ambientes do usuário, faz-se necessário a criação de um conjunto de *plug-ins* para ambientes como *Eclipse* e *Word*. Parte deste trabalho já está em andamento com a construção de duas ferramentas, denominadas: *Basic Asset Retrieval Tool* (*B.A.R.T*) - que envolve a construção de um *plug-in* para o Elipse, e o *Fast Reuse Environment for Office* (*FREvO*) que trata a construção de um *plug-in* para o *Microsoft Word*;
- **Busca ativa:** uma vez integrado ao ambiente do usuário, é extremamente interessante que o repositório suporte a abordagem de busca ativa;
- **Linha de produto:** um trabalho completo de definição de um ambiente de reuso está em construção e para isso será necessário criar uma arquitetura de linha de produto para o repositório proposto. Isto possibilitará a customização do mesmo de acordo com as necessidades específicas de cada contexto onde será implantado;
- **Modelo de Negócio:** afim de permitir que o repositório possa ser utilizado em um contexto mais abrangente de compartilhamento de componentes entre empresas, faz-se necessário a definição de um modelo de negócio que suporte a compra e a venda de componente.

5.4 Conclusão

Empresas estão cada vez mais buscando produzir produtos com um menor custo e tempo e com uma maior qualidade. Repositórios de reuso apropriados podem tornar esses objetivos mais facilmente alcançáveis. Porém, as soluções existentes na academia e na indústria não suportam eficientemente os papéis que um repositório de reuso deveria desempenhar.

Neste sentido, esta dissertação apresentou a especificação, o projeto e a implementação de um repositório de reuso a ser aplicado em fábricas de software. Este repositório foi o resultado de um trabalho de pesquisa desenvolvido em conjunto com a indústria e atualmente se encontra em produção em um contexto com mais de 1.500 desenvolvedores acessando diretamente o repositório corporativo.

Apêndice A -Modelo de ativo

Este apêndice apresenta parte dos detalhes do modelo de classes adotado internamente na estrutura do repositório proposto. A Figura A.1 apresenta o modelo especificado em UML.

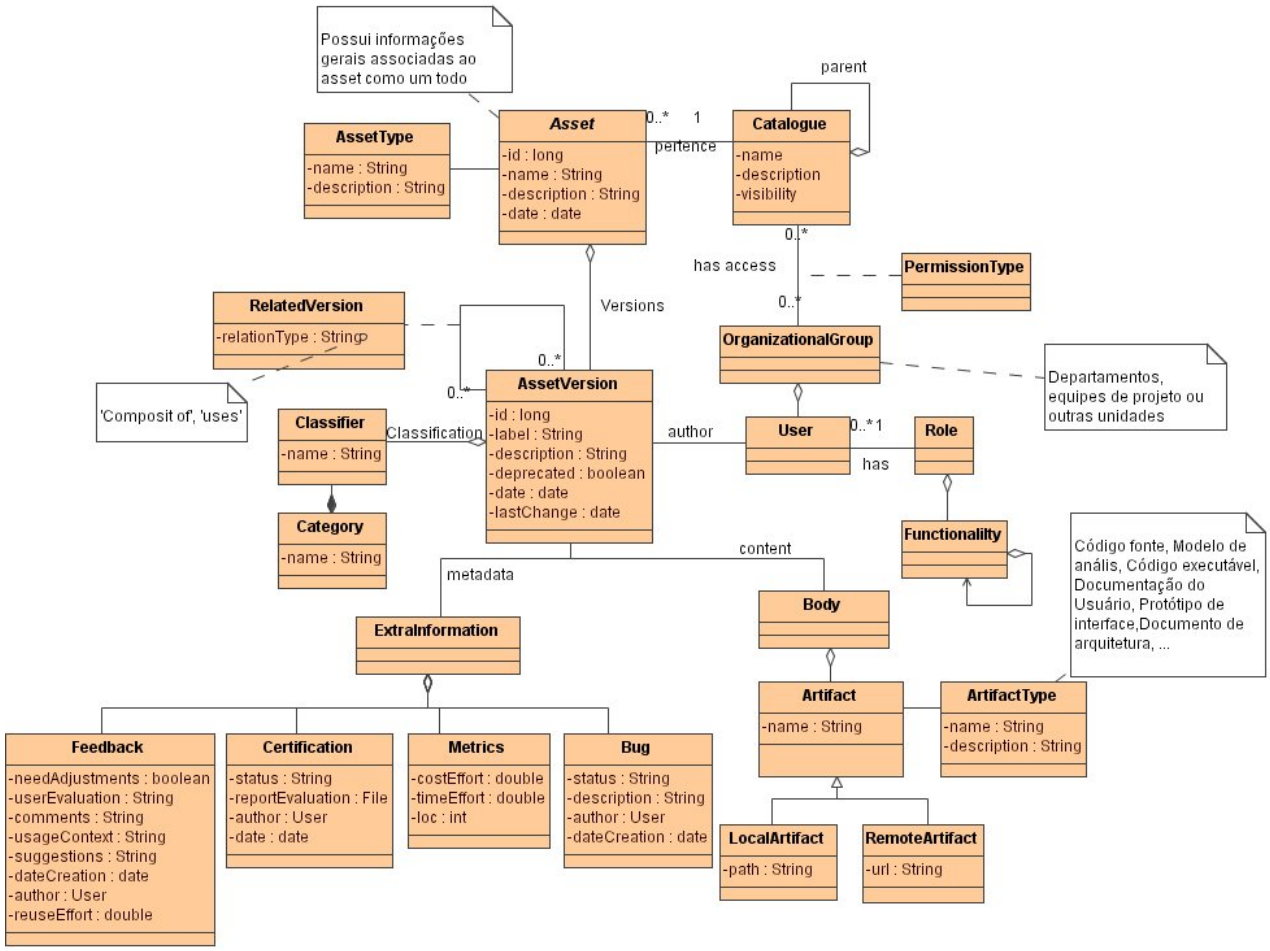


Figura A.1 – Modelo de classes do repositório

B Apêndice B- Exemplos de Relatórios

Este apêndice apresenta alguns exemplos de relatórios que podem ser considerados na implementação de um repositório de ativos que siga a especificação apresentada nesta dissertação. Estes relatórios são estratégicos para a empresa e permitem, entre outras coisas, obter uma visão geral de como o repositório está sendo utilizado e qual o nível de colaboração de seus usuários. De uma maneira geral, podemos dividi-los nos seguintes grupos:

- **Relatórios de produção**: inclui relatórios relacionados ao processo de produção dos *ativos*, tais como, usuários e grupos da organização que mais publicaram *ativos* no repositório;
- **Relatórios de consumo**: inclui relatórios relacionados ao processo de consumo dos *ativos* como, por exemplo, artefatos mais baixados, tipos de *ativos* mais baixados, etc;
- **Relatórios gerais**: incluem outros relatórios como *ativos* mais bem avaliados, nível de certificação dos *ativos* disponíveis, usuários que mais acessaram o repositório, entre outros.

A seguir, para um maior entendimento, são apresentados exemplos de saída de possíveis relatórios de cada um dos grupos acima. É importante ressaltar que são exemplos ilustrativos e que não representam exatamente o mesmo formato de informações e *layout* utilizados nos relatórios implementados.

1. Relatórios de Produção

1.1 Principais grupos organizacionais produtores de ativos

Este relatório lista os ativos produzidos por grupos organizacionais em um dado período. Neste contexto, grupos organizacionais representam unidades da organização que produzem *ativos*. Geralmente constituem equipes de projetos, mas podem representar também departamentos ou até unidades de negócio em uma fábrica de software.

Período: 13/01/2005 – 30/02/2005		
Grupo Organizacional	**Ativo publicado – versão**	**Data de publicação**
Grupo A	Controle de Acesso – 1.0	13/01/2005 - 07:34:36
	Hibernate – 2.0	14/01/2005 - 09:38:36
	Scheduler – 3.0	30/01/2005 - 05:34:36
	Transaction Manager – 1.0	23/01/2005 - 06:33:36
Total de publicações (Grupo A): 4		
Grupo B	Config, Manager – 2.0	13/01/2005 - 07:34:36
	Config, Manager – 2.1	14/01/2005 - 09:38:36
	Config, Manager – 3.0	30/01/2005 - 05:34:36
Total de publicações (Grupo B): 3		
Grupo C	Tag lib – 1.0	13/01/2005 - 07:34:36
	Struts – 1.1	14/01/2005 - 09:38:36
Total de publicações (Grupo C): 3		

1.2 Principais usuários produtores de ativo

O mesmo tipo de relatório pode ser aplicado para identificar os usuários que mais publicaram ativo no repositório.

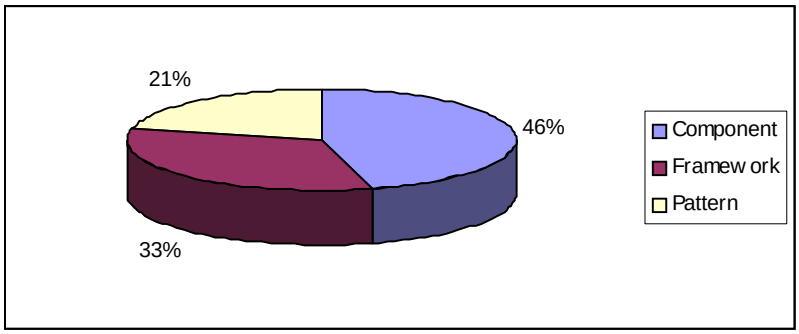
Período: 12/01/2005 – 14/02/2005		
Usuário	**Ativo publicado - versão**	**Data de publicação**
User 1	Controle de Acesso – 1.0	12/01/2005 - 07:34:36
	Hibernate – 2.0	13/01/2005 - 09:38:36
	Scheduler – 3.0	14/01/2005 - 05:34:36
	Transaction Managar – 1.0	14/01/2005 - 06:33:36
Total de publicações (User 1): 4		
User 2	Config, Manager – 2.1	12/01/2005 - 09:38:36
	Tag lib – 1.0	13/01/2005 - 07:34:36
	Struts – 1.1	14/01/2005 - 09:38:36
Total de publicações (User 2): 2		
User 3	Config, Manager – 2.0	13/01/2005 - 07:34:36
	Config, Manager – 2.1	14/01/2005 - 09:38:36
Total de publicações (User 3): 2		

2. Relatórios de Consumo

Tipos de ativo mais baixados

Fornece uma idéia dos tipos de ativo mais procurados em um dado período.

Período: 13/01/2005 – 14/01/2005	
Tipo de ativo	**Downloads efetuados**
Component	49
Framework	35
Pattern	23



Download de artefatos

Apresenta informações a respeito da utilização dos artefatos disponibilizados pelo repositório.

Período: 13/01/2005 – 14/01/2005

Ativo		**Downloads efetuados**
Scheduler – 2.0		
Artefatos	Código Fonte.zip	20
	Modelo de classes	6
	GuiaUsuário.zip	4
Total (Download/artefato): 10		**30**
Controle de acesso – 1.0		
Artefatos	Código Fonte.zip	3
	Modelo de classes	3
Total (Download/artefato): 3		**6**

3. Relatórios Gerais

Assets mais bem avaliados

O feedback fornecido pelo usuário pode ser utilizado para gerar relatórios que apresentem o nível geral de satisfação dos usuários.

Período: 13/01/2005 – 14/01/2005		
Ativo		**Média de satisfação geral**
Controle de acesso		
Versões	2.0	★★★★★
	1.0	★★★☆☆
Média		★★★★☆
Scheduler		
Versões	2.0	★★★★★
	2.1	★★★★★
	1.0	★★☆☆☆
Média		★★★★☆

Nível de certificação dos componentes

Este relatório fornece uma idéia da qualidade dos componentes publicados por grupos organizacionais. Mais detalhes a respeito do processo de certificação inicialmente adotado na solução pode ser visto no Apêndice C.

Período: 13/01/2005 – 20/01/2005		
Grupo Organizacional		**Nível de certificação**
Grupo A		
Ativos	Scheduler 2.1	
	Tag lib 2.0	
	Pool de conexões 1.1	
Total de versões certificadas		**2**
Grupo B		
Versões	Controle de acesso 2.0	
	Gerenciador de transações 2.0	
	Gerador de Senhas 1.0	
Total de versões certificadas		**1**

Legenda:		
	- Certificado;	- Necessita melhorias
	- Em avaliação;	- Rejeitado;

Retorno do investimento

A partir das informações de custo para desenvolvimento do *ativo* (métricas fornecidas pelo produtor do ativo) juntamente com as informações do custo necessário para reutilizar o ativo (informações fornecidas no *feedback* de quem reusou), podemos estimar o retorno de investimento dos *ativos* do repositório.

Ativo	versão	Utilizações	Investimento (R$)	Economia (R$)	Saldo (R$)
Gerenciador de Questões	1.0	1	8.000,00	2.000,00	-6.000,00
Controle de Acesso	1.0	6	11.500,00	43.200,00	31.700,00
Total			19.500,00	45.200,00	25.700,00

Vale ressaltar que o resultado deste relatório é apenas um indicativo e o nível de confiança de suas informações está totalmente atrelada a um uso efetivo da ferramenta.

Apêndice C - Processo de Certificação

Este apêndice apresenta o processo de certificação inicialmente adotado na solução implementada. É importante ressaltar que esse processo foi voltado, inicialmente, para a certificação de componentes de software. Ou seja, ativos que de alguma forma podem ser executados [Ezran et al., 2001]. Por esta razão, será usado neste apêndice o termo 'componente de software' para referenciar os ativos do repositório que podem ser certificados.

Características consideradas no processo de certificação

Dentre os diversos aspectos relacionados à qualidade do componente de software, os usuários e desenvolvedores estão principalmente preocupados com 2 aspectos [Keil & Tiwana, 2005]:

- **Funcionalidade**
- **Confiabilidade**

Baseado nisso, os aspectos que, primeiramente, foram considerados para obter um nível de qualidade nos componentes disponibilizados no repositório foram as duas características de qualidade apresentadas. Assim, a Tabela C.8.1 apresenta as técnicas que foram utilizadas para cada característica.

Características	Técnicas		
Funcionalidade	Teste funcional (caixa preta)	Inspeção da documentação do componente (ver lista abaixo)	Teste completo e inspeção de código (caixa branca)
Confiabilidade	Facilidades da linguagem de programação	Análise da tolerância à falhas e da manipulação de erros.	Análise da confiabilidade do componente (tempo q ocorre uma falha e tempo p/ se recuperar)

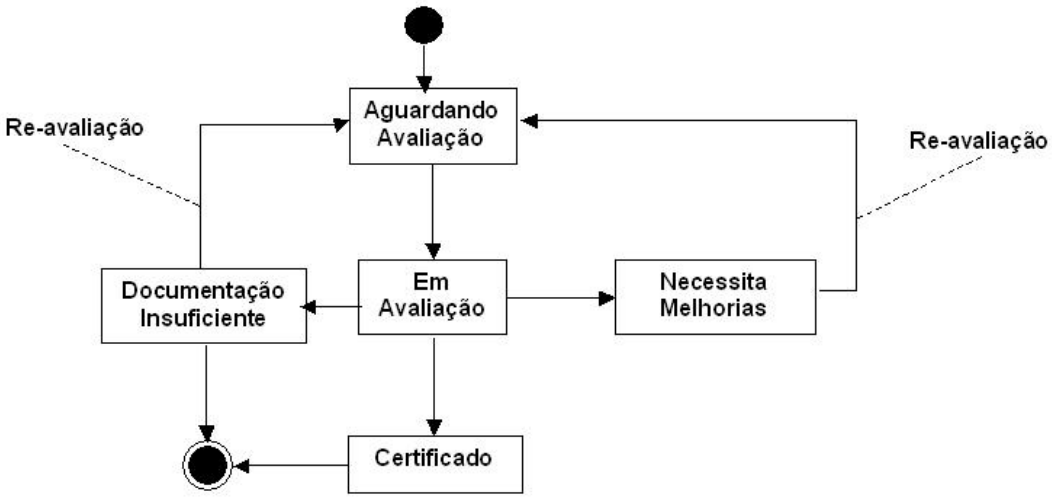
Tabela C.8.1 – Características e técnicas utilizadas no processo de certificação

Documentos necessários para a avaliação:

- Documento de Requisitos;
- Documento de Casos de Uso;
- Diagrama de Classes (caso tenha maiores diagramas também);
- Modelo de Dados;

- Código Fonte;
- Unidades de Teste;
- Guia de Uso do Componente (Tutorial? Help system?);
- API;
- Aplicação(ões) que o componente está sendo utilizado; e Exemplos de utilização do componente.

A seguir, é apresentado um diagrama com os estados que um componente pode assumir, a partir de seu *upload* no repositório.



Estados de um componente

1. **Aguardando Avaliação**: Fase na qual o ativo ainda não foi avaliado ou recentemente foi submetido a uma nova avaliação por ter apresentado problemas na última avaliação;

2. **Em Avaliação**: Fase na qual o avaliador utiliza as técnicas descritas na tabela acima e realiza a avaliação do componente. Os passos para realizar o processo de avaliação do componente são descritas na próxima Seção. Esta fase gera um

documento chamado “Avaliação do Componente”, onde serão descritos os testes realizados, os resultados e algumas considerações serão delineadas;

3. **Necessita Melhorias**: Esta fase conterá um documento relatando quais são as deficiências do componente e quais são os pontos que ele pode ser melhorado. Caso o cliente realize as modificações necessárias, este componente poderá ser reavaliado para que seu status seja atualizado;

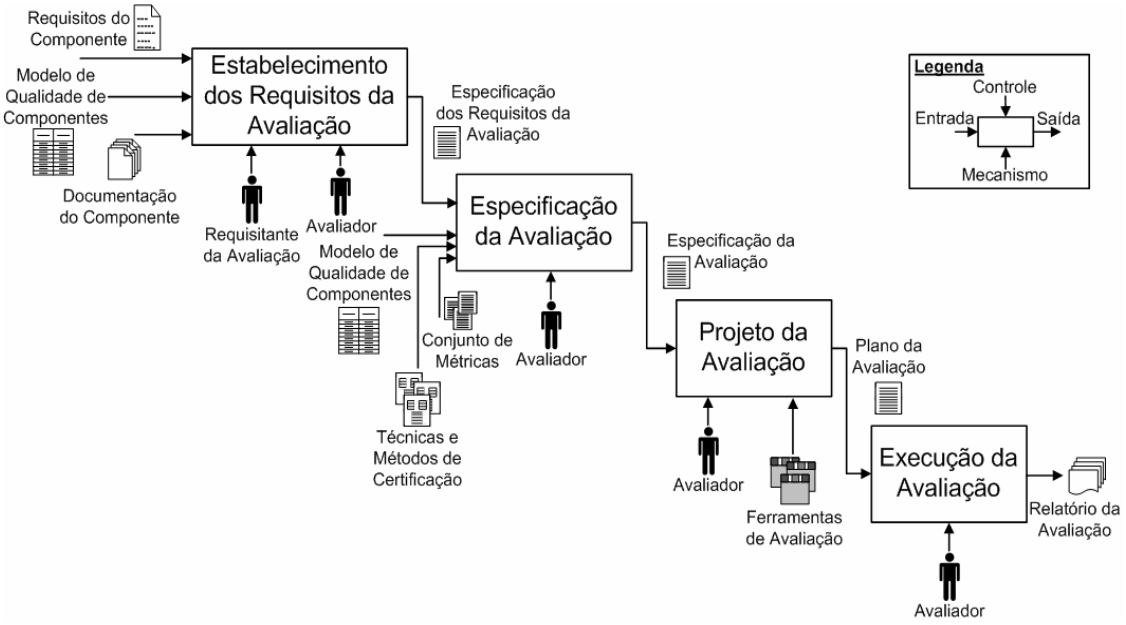
4. **Certificado**: Esta fase conterá um documento relatando quais testes, técnicas, métodos, ferramentas foram utilizadas para realizar a avaliação do componente. Ainda é apresentado o resultado dos testes e possíveis melhorias que poderiam ser realizadas (sugestões);

5. **Documentação Insuficiente**: Esta fase conterá um documento relatando quais são os documentos que estão faltando para que o avaliador possa realizar a avaliação do componente (definir quais são os documentos mínimos que devem estar disponíveis para a avaliação).

Passos do processo de avaliação

A processo de avaliação é executado em 4 (quatro) passos:

- Estabelecimento dos Requisitos da Avaliação;
- Especificação da Avaliação;
- Projeto da Avaliação;
- Execução da Avaliação.



1. Estabelecimento dos Requisitos da Avaliação

- Definir o time para realizar a avaliação;
- Definir os objetivos e o escopo da avaliação
- Definir os riscos associados;
- Analisar os atributos de qualidade e requisitos do sistema;
- Definir os requisitos do COTS a ser avaliado em conjunto com o requisitante da avaliação (ou podem-se definir níveis no modelo ou quais atributos serão considerados.);
- Deve haver uma descrição dos requisitos que serão submetidos à avaliação;
- Ainda, podem existir requisitos que o modelo de qualidade de componentes não cobre, assim, deve ser feita uma referência à literatura conceituada que a definida, ficando claro o entendimento de tais requisitos identificado;
- Devem ser informadas também quais informações o sistema e/ou o componente deve conter para que os requisitos levantados possam ser avaliados;
- Identificar quais atributos/requisitos do sistema interfere na avaliação do componente (Análise de Aplicabilidade?):
  (i) Restrições com a Arquitetura/Interface
  (ii) Restrições da Linguagem de Programação
  (iii) Ambiente Operacional
- Elaborar o documento com os requisitos que serão avaliados.

2. **Especificação da Avaliação**

- Baseado nos requisitos e atributos de qualidade definidos no passo anterior, este passo deve (o melhor possível, para garantir a repetibilidade e reprodutividade do processo de avaliação):
  (i) Correlacionar os requisitos com os atributos de qualidade do modelo de qualidade de componentes;
  (ii) Especificar as técnicas para certificação que será utilizada para avaliar cada característica/atributo de qualidade. A escolha das técnicas vai depender do nível de confidencia esperado pelo resultado da avaliação;
  (iii) Especificar as métricas que serão utilizadas para mensurar os atributos;
  (iv) Especificar as métricas para medir o quão eficiente o processo de avaliação foi. Isso pode ser armazenado em uma base de conhecimento para futuras avaliações (realizar comparações);
  (v) Verificar a lista de componentes, módulos e/ou sistemas que o componente a ser avaliado depende. Assim, eles devem ser especificados nesse doc., descritos, nível de dependência, etc.; e
  (vi) Elaborar o documento de Especificação da Avaliação, verificando a especificação produzida em relação aos requisitos de avaliação definidos no passo anterior.

3. Projeto da Avaliação

- Obter a documentação do componente e do sistema que o componente está implantado;
- Desenvolver ferramentas para avaliar os atributos de qualidade, caso necessário. Ainda, definir se alguma ferramenta presente na literatura e/ou mercado poderá ser utilizada;
- Conhecimento de técnicas, linguagens de programação, ambientes e recursos requeridos para avaliar o componente;
- Fazer cronograma para realizar a certificação;
- Descrever os métodos e técnicas que serão adotados para avaliar cada requisito/atributo de qualidade definido anteriormente. Como cada técnica/método deve ser utilizado na avaliação. Documentar cada técnica e método;
- Descrever as métricas que serão utilizadas para mensurar o quão eficiente foi o processo, armazenando tais informações para comparações futuras. Descrever como cada métrica realiza a medição de cada característica de qualidade;
- Elaborar o Plano da Avaliação, contendo todas as etapas descritas aqui (Composto basicamente de 2 partes: documentação dos métodos de avaliação e cronograma das ações do avaliador.).

4. Execução da Avaliação

- Executar os métodos e técnicas descritas no Plano de Avaliação, utilizando as métricas descritas nesse doc. também para medir as informações geradas nesta fase;
- Analisar os resultados obtidos e gerar um relatório sobre onde pode-se obter melhorias no componente (recomendações). Armazenar isso em uma base para comparações em futuras avaliações;
- Deve-se considerar um ambiente de execução e um sistema para avaliar o componente. Com isso, o componente será certificado neste determinado ambiente.
- Elaborar um documento de Avaliação do Componente, onde serão descritos os testes realizados e serão apresentados os resultados e algumas considerações serão delineadas:
    i. Descrição do produto – componente;
    ii. Descrição das funções;
    iii. Requisitos de hardware e software;
    iv. Informações sobre a configuração do componente, como deve ser realizada;
    v. Interface com outros componentes;
    vi. É oferecido suporte?;
    vii. É oferecido manutenção?;
    viii. Devem conter declarações sobre funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade;
    ix. Verificar todos os atributos de qualidade do modelo de qualidade, neste caso do modelo de qualidade de componentes;

eferências

[Abd-Allah, 1994] A. A. Abd-Allah. **Architecture Description Languages**. 1994.

[Almeida et al., 2005] E. Almeida, A. Alvaro, S. Meira, **RiSE Project: Key Developments in the Field of Software Reuse**, 15th PhDOOS Workshop Glasgow, Scotland, 2005.

[Almeida et al., 2005b] E. Almeida, A. Alvaro, D. Lucrédio, V. Garcia, S. Meira, **A Survey on Software Reuse Processes**, In the IEEE International Conference on Information Reuse and Integration, Las Vegas, USA, 2005.

[Alvaro et al., 2005] A. Alvaro, E. Almeida, S. Meira, **Software Component Certification: A Survey**, 31st IEEE EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Component-Based Software Engineering Track, Portugal, IEEE Press, 2005.

[Aquino, 2002] G. S. Aquino, **Development of Web-based Java Systems: Frameworks, Patterns and Guidelines for Presentation Layer,** Master Dissertation, Federal University of Pernambuco, Recife, Brazil, 2002.

[Apperly, 2001] H. Apperly, **Configuration Management and Component Libraries**, In: Component-Based Software Engineering: Putting the Pieces Together, Addison Wesley, 2001, pp. 513-526

[Baeza-Yates & Ribeiro-Neto, 1999] R. Baeza-Yates, B. Ribeiro-Neto, **Modern Information Retrieval**, ISBN: 020139829X, 1st edition, ACM Press, 1999.

[Barbar et al., 2004] M. A. Babar, L. Zhu, R. Jeffery, **A Framework for Classifying and Comparing Software Architecture Evaluation Methods**, In: Proceedings of the Australian Software Engineering Conference, 2004.

[Basili, 1991] V.R. Basili, H. D. Rombach, **Support for Comprehensive Reuse**, Software Engineering Journal, Special issue on software process and its support, Vol. 06, No. 05, April, 1991, pp. 306-316.

[Bass et al., 1998] L. Bass, P. Clements, R. Kazman, **Software Architecture in Practice**, Addison-Wesley, 2003.

[Banker et al, 1993] R. D. Banker, R. J. Kauffman, D. Zweig, **Repository Evaluation of Software Reuse**, In: IEEE Transaction on Software Engineering, Vol. 19, No 4, April, 1993, pp. 379-389

[Bauer & King, 2005] C.Bauer, G. King, **Hibernate in Action**, Manning Publications Co., ISBN 1932394-15-X, 2005

[Blaha et al, 1998] M. Blaha, D. LaPlant , E. Marvak, **Requirements for Repository Software**, In: IEEE Software, 1998

[Blechar, 2005] M. J. Blechar, **Magic Quadrant for Metadata Repositories,** Gartner Research, ID Number: G00129274, Publication Date: 30 June 2005.

[Booch et al., 1999] G. Booch, I. Jacobson,and J. Rumbaugh, **The Unified Modeling Language Reference Manual**, Addison-Wesley, 1999.

[Burégio et al., 2006] V. A. A. Burégio, E. C. Santos, E. S. Almeida, S. R. L. Meira, **Industrial Experiment in Software Architecture Evaluation,** 2006. (In evaluation).

[Casanova el al., 2003] M. Casanova, R. V. D. Straeten and V. Jonckers, **Supporting Evolution in Component-Based Development using Component Libraries**, In: Proceedings of the 7th European Conference on Software Maintenance and Reengineering (CSMR'03), 2003.

[Clements, 1996] P. C. Clements, **A Survey of Architecture Description Languages**, In: Eighth International Workshop on Software Specification and Design, Germany, March 1996

[Clements et al., 2002a] P. Clements, R. Kazman, and M. Klein, **Evaluating Software Architectures: Methods and Case Studies**, Addison-Wesley, 2001.

[Clements, 2002b] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, J. Stafford, **Documenting Software Architectures: Views and Beyond**, Addison-Wesley, 2002, pp. 560.

[Dobrica & Niemela, 2002] L. Dobrica and E. Niemela, **A Survey on Software Architecture Analysis Methods**, IEEE, 2002

[Dogru & Tanik, 2003] A. Dogru, M. Tanik, **A Process Model for Component-Oriented Software Engineering**, IEEE Software, vol. 20, no. 2, pp. 34-41,2003.

[D'Souza & Wills, 1999] D.F. D'Souza, A. C. Wills, **Objects, Components, and Frameworks With UML: The Catalysis Approach**, Addison-Wesley Object Technology Series, 1999.

[Etzkorn & Davis, 1997] L. H. Etzkorn, C. G. Davis, **Automatically Identifying Reusable OO Legacy Code**, Computer, IEEE, October, 1997.

[Ezran el al., 2002] M. Ezran, M. Morisio, C. Tully, **Practical Software Reuse**, Springer-Verlag, 1st Edition, ISBN: 1852335025, 2002.

[Feldman et al., 2000] R. L. Feldman, M. Frey, M. Habetz, M. Mendoça, **Applying Roles in Reuse Repositories**, 2000, In: Proceedings of the Twelfth International Conference on Software Engineering and Knowledge Engineering

[Favaro, 1991] J. Favaro, **What price reusability? A case study**, Ada Letters (Spring 1991): 115-124.

[Fenton, 1991] N. Fenton, **Software Metrics: A Rigorous Approach**, Chapman & Hall, London, 1991.

[Fichman & Kemerer, 2001] R. Fichman, C. Kemerer, **Incentive Compatibility and Systematic Software Reuse**, Journal of Systems and Software, 57(1),2001.

[Fischer, 1987] G. Fischer, **A Critic for LISP**, in J. McDermott (ed.), Proceedings of the 10th International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Altos, USA, pp. 177184, 1987.

[Frakes & Gandel, 1990] W. Frakes and P. Gandel, **Representing reusable software**. Information and Software Technology, 1990.

[Frakes & Fox, 1995] W.B. Frakes, C.J. Fox, **Sixteen Questions about Software Reuse**, Communications of the ACM, Vol. 38, No. 06, June, 1995, pp. 75-87.

[Frakes & Succi, 2001] W. Frakes, G. Succi, **An Industrial Study of Reuse, Quality, and Productivity**, Journal of Systems and Software, Volume 57, Issue 2, No. 15, pp. 99-106, 2001.

[Frakes & Terry, 1996] W. Frakes, C. Terry, **Software reuse: metrics and models**, ACM Computing Surveys, Vol. 28, No. 02, ACM Press, 1996.

[Fowler, 2004] M. Fowler, **Inversion of Control Containers and the Dependency Injection Pattern**, 2004, Available on: http://martinfowler.com/articles/injection.html. Retrieved on 2005-11-15

[Gaffney & Durek, 1989] J. Gaffney, T. Durek, **Software reuse - key to enhanced productivity: some quantitative models**, Information and Software Technology 31 (5), pp. 258-267, 1989.

[Gamma et al., 1994] E. Gamma, R. Helm, R. Johnson, J. Vlissides, **Design Patterns: Elements of Reusable Object-Oriented Software**, AddisonWesley Professional, ISBN 0201633612, 1st Edition, 1994.

[Garcia et al., 2006] V. Garcia, D. Lucrédio, F. Durao, E. Santos, E. Almeida, R.Fortes, S. Meira, **From Specification to the Experimentation: A Software Component Search Engine Architecture**, Proceedings of the 9th International SIGSOFT Symposium on Component-Based Software Engineering,Vasteras, Sweden, 2006.

[Garcia et al., 2006b] V. Garcia, E. Almeida, S. Meira, D. Lucrédio, R. Fortes, **A Roadmap on Software Reuse Environments and Tools**, 21st IEEE/ACM International Conference on Automated Software Engineering, Tokyo, Japan, (in evaluation) 2006.

[Garlan & Shaw, 1993] D. Garlan & M. Shaw, **An Introduction to Software Architecture**, Advances in Software Engineering and Knowledge Engineering, Volume I, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.

[Ghezzi et al., 2002] C. Ghezzi, M. Jazayeri, D. Mandrioli, **Fundamentals of Software Engineering**, Second Edition, ISBN 0-13-305699-6, 2002.

[Glass, 1998] R. Glass, **Reuse: What's Wrong with this Picture?**, IEEE Software,Vol. 15, No. 02, 1998.

[Gospodnetic & Hatcher, 2004] O. Gospodnetic, E. Hatcher, **Lucene in Action**, Manning Publications Co., ISBN 1-932394-28-1, 2004.

[Griss, 1994] M. Griss, **Software reuse experience at Hewlett-Packard**, Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, 1994.

[Griss et al., 1995] M. Griss, M. Wosser, S. Pfleeger, **Making Software Reuse Work at Hewlett-Packard**, IEEE Software, Vol. 12, No. 01, 1995.

[Hass, 2003] A. M. J. Hass, **Configuration Management Principles and Practice**, Addison Wesley, 2003.

[Henninger, 1997] S. Henninger. **An evolutionary approach to constructing effective software reuse repositories**. ACM Transactions on Software Engineering and Methodology, 6(2):111–140, 1997.

[Husted et al., 2003] T. Husted, C. Dumoulin, G. Franciscus, D. Winterfeldt, **Struts in Action**, Manning Publications Co., ISBN 1-930110-50-2, 2003

[Inoue et al., 2003] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matusushta, an S. Kusumoto, **Componet rank: relative significance rank for software component search**. In Proceeding of the 25 th International Conference on Software Engineeing, pages 14-25, 2003

[Isakowitz & Kauffman, 1996] T. Isakowitz an R. J. Kauffman, **Supporting search for reusable software objects**, IEEE Trans. on Software Engineering, 1996

[John & Spiros, 1996] H. John, G. Spiros, **Reuse Concepts and a Reuse Support Repository**, IEEE, 1996.

[Kazman et al., 1999] R. Kazman, M. Barbacci, M. Klein, S. Carriere, and S. Woods, **Experience with performing architecture tradeoff analysis**, In Proceedings of the 21st International Conference on Software Engineering (ICSE 21), Los Angeles, CA, pp. 54–63, 1999.

[Joos, 1994] R. Joos, **Software Reuse at Motorola**, IEEE Software, Vol. 11, No.05, 1994.

[Kazman et al., 2000] R. Kazman, M. Klein, and P. Clements, P, **ATAM: A method for architecture evaluation**, CMU/SEI- 2000-TR-004, Software Engineering Institute, Carnegie Mellon University.

[Kazman et al., 2005] R. Kazman, L. Bass, M. Klein, T. Lattanze, L. Northrop, **A Basis for Analyzing Software Architecture Analysis Methods**, In: Software Quality Journal, 13, 329–355, 2005

[Kazman et al., 1994] R. Kazman, G. Abowd, L. Bass and M. Web, **SAAM: A method for analyzing the properties of software architectures**, In Proceedings of the 16th International Conference on Software Engineering. Sorrento, Italy, May 16–21, Los Alamitos, CA: IEEE Computer Society, pp. 81–90.

[Kernebeck, 1997] U. Kernebeck, **Component libraries for Software re-use**, Microprocessors and Microsystems - Published by Elsevier Science B. V., 1997

[Krueger, 1992] C.W. Krueger, **Software Reuse**, ACM Computing Surveys, Vol. 24, No. 02, June, 1992, pp. 131-183.

[Larman, 2003] G. Larman, **Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development** (2nd Edition), By Bookman Companhia Ed., 2003.

[Lim, 1994] W. Lim, **Effects of Reuse on Quality, Productivity, and Economics**,IEEE Software, Vol. 11, No. 5, 1994.

[Lim, 1998] W. Lim, Managing Software Reuse, Prentice Hall, 1998, ISBN 0135523737.

[Lynex & Layzell,1998] Andy Lynex, Paul Layzell, **Organizational Considerations for Software Reuse**, 1998.

[Lucredio et al., 2004] D. Lucrédio, A. Prado, E. Almeida, A Survey on Software Components Search and Retrieval, In the 30th IEEE EUROMICRO Conference, Component-Based Software Engineering Track, Rennes, France, IEEE Press, 2004.

[Maranzano et al., 2005] J. F. Maranzano, S. A. Rozsypal, G. H. Zimmerman, G. W. Warnken, P. E. Wirth, D. M. Weiss, **Architecture Reviews: Practice and Experience**, Published by the IEEE Computer Society, March/April 2005.

[Mascena et al., 2005] J. Mascena, E. Almeida, S. Meira, **A Comparative Study on Software Reuse Metrics and Economic Models from a Traceability Perspective**, IEEE Information Reuse and Integration, Las Vegas, USA, 2005.

[Mascena et al., 2006] J. Mascena, **ADMIRE: Asset Development Metrics-based Integrated Reuse Environment**, MSc dissertation, Federal University of Pernambuco, Brazil, 2006.

[McIlroy, 1968] M.D. McIlroy, **Mass Produced Software Components**, NATO Software Engineering Conference Report, Garmisch, Germany, October, 1968, pp. 79-85.

[Medvidovic & Taylor, 2000] N. Medvidovic and R. Taylor, **A Classification and Comparison Framework for Software Architecture Description Languages,** IEEE Transactions on Software Engineering, vol. 26, pp. 70-93, 2000.

[Mendes, 2006] R. C. Mendes. **An Automatic Asset Extractor Engine**, Master Dissertation Proposal, Federal University of Pernambuco, Recife, Brazil, 2006 (to apper).

[Mili et al., 1997] R. Mili, A. Mili, R. T. Mittermeir, **Storing and Retrieving Software Components: A Refinement Based System**, IEEE Transactions on Software Engineering, Vol. 23, 1997, pp. 445-460.

[Mili et al., 1998] A. Mili, R. Mili, R. Mittermeir, **A Survey of Software Reuse Libraries**, Annals Software Engineering, Vol. 05, 1998, pp. 349–414.

[Morisio, 2002] M. Morisio, M. Ezran, C. Tully, **Success and Failure Factors in Software Reuse** , IEEE Transactions on Software Engineering, Vol. 28, No. 04, April, 2002, pp. 340-357.

[Pan et al., 2004] Y. Pan, L. Wang, L. Zhang, **Relevancy Based Semantic Interoperation of Reuse Repositories**, SIGSOFT'04/FSE-12, 2004, pp. 211-220.

[Perry et al., 2000] D. Perry, A. Porter, L. Votta, **Empirical Studies of Software Engineering: A Roadmap**, The Future of Software Engineering, Anthony Finkelstein (Ed.), ACM Press, ISBN 1-58113-253-0, 2000.

[Podgurski & Pierce, 1993] A. Podgurski, L. Pierce, **Retrieving Reusable Software by Sampling Behavior**, ACM Transactions on Software Engineering and Methodology 2(3), pp. 286303, 1993.

[Poulin, 1995] J. Poulin, **Populating Software Repositories: Incentives and Domain-Specific Software**, Journal of Systems and Software, Vol. 30, No. 3, pp. 187-195, 1995.

[Poulin, 1997] J. Poulin, **Measuring Software Reuse**, Addison-Wesley, 1997.

[Prieto-Diaz, 1991] R. Prieto-Diaz. **Implementing faceted classification for software reuse**. Communications of the ACM, 34(5), 1991.

[Prieto-Diaz & Freeman, 1987] R. Prieto-Diaz, P. Freeman, **Classifying Software for Reusability**, IEEE Software, Vol. 04, No. 01, January, 1987, pp. 06-16.

[RAS, 2004] **Reusable Asset Specification**. 2004. Details available from http://www.omg.org/docs/ptc/05-04-02.pdf (last accessed 20 May 2005).

[Raymond, 2001] E. S. Raymond, B. Young, **The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary**, O'Reilly: Sebastopol, CA, 2001.

[RUP, 2006] **Rational Unified Process**. Consultado em 20006, http://www.ibm.com/software/awdtools/rup/

[Sametinger, 1997] J. Sametinger, **Software Engineering with Reusable Components**, Springer Verlag, 1997.

[Seacord et al., 1998] R. C. Seacord, S. A. Hissam, and K. C. Wallnau. **Agora: A search engine for software components**. Technical report, CMU/SEI - Carnegie Mellon University/Software Engineering Institute, 1998.

[Ship, 2004] H. M. L. Ship, **Tapestry in Action**, Manning Publications Co.ISBN 1-932394-11-7, 2004

[Guo & Luqi, 200] J. Guo, Luqi, **A Survey of Software Reuse Repositories**, Proceedings of the Seventh IEEE Conference and Workshop on the Engineering of Computer-Based Systems, April 2000.

[Vanderlei, 2006] T. A. Vanderlei, **A Cooperative Classification Mechanism for Searching and Retrieving Software Components**, Master Dissertation, Federal University of Pernambuco, Recife, Brazil, p. 129, 2006.

[Voas, 1998] J. M. Voas, **Certifying Off-the-Shelf Software Components**, IEEE Computer, vol. 31, pp. 53-59, 1998.

[Mingins & Schmidt, 1998] C. Mingins and H. Schmidt, **Providing Trusted Components to the Industry**, IEEE Computer, vol. 31, pp. 104-105, 1998.

[Wallnau, 2003] K. C. Wallnau**, A technology for predictable assembly from certifiable components**, Technical report, CMU/SEI - Carnegie Mellon University/Software Engineering Institute, 2003.

[Ye, 2001] Y. Ye, **Supporting Component-Based Software Development with Active Component Repository Systems**, PhD thesis, University of Colorado, Boulder, USA, 2001.

[Zhuge, 2000] H. Zhuge, **A problem-oriented and rule-based component repository**, The Journal of Systems and Software, 50:201–208, 2000.

Dissertação de Mestrado apresentada por **Vanilson André de Arruda Burégio** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Specification, Design and Implementation of a Reuse Repository"**, orientada pelo **Prof. Silvio Romero de Lemos Meira** e aprovada pela Banca Examinadora formada pelos professores:

Prof. André Luis de Medeiros Santos
Centro de Informática / UFPE

Prof. Ricardo Massa Ferreira Lima
Escola Politécnica de Pernambuco / UPE

Prof. Silvio Romero de Lemos Meira
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 5 setembro de 2006.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.