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

DÊNIS PAIVA RAMOS

AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE (ADS) & FERRAMENTAS CASE: IMPORTÂNCIA E APLICAÇÕES.

Belo Horizonte
2011

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

AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE (ADS) & FERRAMENTAS CASE: IMPORTÂNCIA E APLICAÇÕES.

por

DÊNIS PAIVA RAMOS

Monografia de Final de Curso

Prof. Ângelo de Moura Guimarães
Orientador

Belo Horizonte
2011

DÊNIS PAIVA RAMOS

AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE (ADS) & FERRAMENTAS CASE: IMPORTÂNCIA E APLICAÇÕES.

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

Área de concentração: Análise de Sistemas.

Orientador: Prof. Ângelo de Moura Guimarães.

Belo Horizonte
2011

AGRADECIMENTOS

Ao Deus da Vida.

Aos meus professores.

Aos meus pais.

Aos meus amigos e confrades.

“Sociedades e instituições profissionais têm um importante papel a desempenhar, definindo padrões de ética .”

(Sommerville)

“O homem livre, no que pensa menos é na morte, e a sua sabedoria é uma meditação, não da morte, mas da vida.”

(Baruch Spinoza)

RESUMO

O principal objetivo do presente trabalho é apresentar conceitos importantes, as suas aplicações e importâncias no desenvolvimento de um software, tais como: Ambiente de Desenvolvimento de Software, Processo de Desenvolvimento de Software, Ferramentas CASE e outros. Com isso, perceber a importância destas ferramentas para um engenheiro de software e suas aplicações diz respeito às novas posições no mercado atual de desenvolvimento de software. É necessário conhecer os ambientes, os processos e as ferramentas, pois são prioridades para um engenheiro de software que deseja garantir maior qualidade e rapidez no desenvolvimento de um software, facilitando e evitando maiores problemas na etapa de manutenção. Objetivar e automatizar o apoio ao processo de desenvolvimento de software é meta específica nesta área para as empresas, pois uma garantia de evolução e atualização é necessária neste processo. Assim, a utilização de Ferramentas CASE e linguagem como a UML no processo de documentação e visão geral do projeto são indispensáveis. Ao final deste trabalho é citada a Ferramenta Eclipse como exemplo de Ferramenta CASE, constituindo de vários *plugins* que contribuem no processo de desenvolvimento de um software. Um elemento de grande importância da utilização destas ferramentas é entender a prioridade da integração das diversas ferramentas que assim oferecem apoio no desenvolvimento de um software.

Palavras-chave: Ferramentas CASE, Ambientes de Desenvolvimento de Software, Processo de Desenvolvimento de Software, Eclipse, UML, Integração.

ABSTRACT

The main objective of this work is to present important concepts, their applications and importance in the development of a software such as: Software Development Environment, Software Development Process, CASE tools and others. Thus, realizing the importance of these tools to a software engineer and applications regarding new positions in the current market of software development. It is necessary to know the environments, processes and tools, as are priorities for a software engineer who wants to ensure higher quality and speed in development of software, making it easy and avoid major problems in the upkeep. Targeting support and automate the process of software development is this area to target specific companies, as a guarantee of progress and update is required in this process. Thus, the use of CASE tools and language such as UML in the process of documentation and overview of the project are indispensable. At the end of this work is cited as an example Eclipse Tool CASE tool, providing a number of plugins that contribute to the process of developing a software. An important element in the use of these tools is to understand the priority of integrating so many tools that provide support in developing a software.

Keywords: CASE Tools, Software Development Environments, Software Development Process, Eclipse, UML, Integration.

LISTA DE FIGURAS

LISTA DE SIGLAS

ACP: Ambientes Centrados em Processo.

ADS: Ambiente de Desenvolvimento de Software.

ADSCP: Ambientes de Desenvolvimento de Softwares Centrados em Processos.

ADSOD: Ambientes de Desenvolvimento de Softwares Orientados a Domínio

ADSOrg: Ambientes de Desenvolvimento de Softwares Orientados à Organização.

CASE: *Computer-Aided Software Engineering* - Engenharia de Software Auxiliada por Computador.

CFG: *Control Flow Graphics* - Gráficos de Fluxo de Controle.

EPL: *Eclipse Public License* – Licença Pública do Eclipse

EMF: *Eclipse Modeling Framework* – Framework de Modelagem do Eclipse.

ES: Engenharia de Software.

IDEs: *integrated development environments* - ambientes integrados de desenvolvimento.

JDT: *Java Development Tools* – Ferramentas de Desenvolvimento Java.

MDA (Model Driven Architecture, Arquitetura Dirigida pelo Modelo).

ODE: *Ontology-based software Development Environment* - Ontologia baseada em Ambiente de Desenvolvimento de Software.

PDE: *Plug-in Development Environment* – Ambiente de Desenvolvimento de Plug-in.

PDS: Processo de Desenvolvimento de Software.

PSM: Platform Specific Model - Modelos Específicos de Plataforma.

PP: Processo Práxis.

PSP: Processo Pessoal de Software.

RTE: *Round-Trip Engineering* - Engenharia de ida e volta.

RUP: *Rational Unified Process*.

SDK: *Eclipse Software Development Kit.*

TSP: *Team Software Process*, Processo de Software para Times.

UML: *Unified Modeling Language* – Linguagem de Modelagem Unificada.

UP: *Unified Process*, Processo Unificado.

SUMÁRIO

1 INTRODUÇÃO

O objetivo deste trabalho é discutir o uso de ferramentas CASE (Computer Aided Software Engineering) necessárias e importantes para um Analista de Sistemas e um Engenheiro de Software - no apoio ao desenvolvimento de sistemas de software, ajudando-os, no intuito de aumentar a segurança e qualidade dos sistemas desenvolvidos. O trabalho destaca também que a integração entre estas ferramentas é fator de suma importância.

O presente trabalho está dividido em quatro capítulos (partes):

Na primeira parte serão abordados os ambientes de desenvolvimento de software, mostrando alguns tipos e especificando a característica e importância de cada um. Em seguida serão mostrados os diversos processos de desenvolvimento de software, como também seus modelos, fases e processos. Por fim, uma pequena abordagem sobre a linguagem UML de modelagem de sistemas, utilizando diagramas.

Na segunda parte, serão tratados especificamente as ferramentas CASE. A integração entre uma ferramenta CASE e um ADS (Ambiente de Desenvolvimento de Software) procura relacionar o conceito de apoio ao desenvolvimento de software. O conceito de ferramenta CASE e sua utilização nas indústrias vem sendo observado por vários analistas de sistemas e já conta com a adesão de muitos engenheiros de software.

Na terceira parte, de forma mais prática, será apresentado como exemplo um caso de ferramenta CASE utilizando o ambiente Eclipse e um plug-in (eUML2) que associa UML e a geração de código Java. A utilização de vários plug-ins é característica no ambiente Eclipse, no qual, para melhor visão da globalidade de um sistema pelo engenheiro de software. Também será apresentado o conceito e a aplicação de engenharia de ida e volta, ou seja, engenharia de produção e engenharia reversa.

Por fim, nas considerações finais será apresentada uma conclusão de todo o trabalho levando em conta os elementos positivos e negativos do desenvolvimento do mesmo. Será traçado também uma análise da utilização das ferramentas especificadas nas partes anteriores e as perspectivas de trabalho para sua melhoria, ou seja, uma visão do que poderia ser mais bem trabalhado e atualizado para seu aperfeiçoamento, levantando as críticas e os elogios obtidos no final do trabalho.

2 AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE E PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

2.1. Ambientes de Desenvolvimento de Software.

O desejo de qualquer empresa ou equipe para desenvolver um software de qualidade garantida e assegurada tem sido um grande desafio dentro da Engenharia de Software (ES). Porém, a utilização de bons Ambientes de Desenvolvimento de Software (ADS) e Processos de Desenvolvimento de Softwares (PDS) podem levar a uma maior elaboração de todo o processo que implicará num apoio de ferramentas especializadas para o efetivo ciclo de vida de um software. Os ADSs podem ser definidos como ambientes/sistemas que irão objetivar o apoio ao processo de software, apoiando cada atividade desse complexo processo, utilizando diversas ferramentas e facilidades integradas. Sabe-se, no entanto, que construir este tipo de ambiente e percurso não é algo tão fácil.

> Cada vez mais engenheiros de software têm sido cobrados para realmente fazerem engenharia do produto de software: planejar, acompanhar, executar e controlar. Cresce, então, a necessidade de ferramentas para apoiar estas tarefas... Neste contexto, é crescente a demanda por Ambientes de Desenvolvimento de Software (ADSs), que buscam combinar técnicas, métodos e ferramentas para apoiar o Engenheiro de Software na construção de produtos de software, abrangendo todas as atividades inerentes ao processo, tais como planejamento, gerência, desenvolvimento e controle da qualidade (MIAN *et al*, 2001, p. 2).

Na ES, como acontece com várias outras áreas da informática, a evolução e atualização de ferramentas é algo da *práxis*, pois é perceptível e necessário nos métodos e nas formas de apoiar o desenvolvimento de software. Exemplo disso temos a tecnologia CASE, na qual veremos no capítulo seguinte deste trabalho. Com o ADS há uma disponibilização de várias ferramentas de apoio ao longo de todo o processo de desenvolvimento, apresentando também uma especificidade de cada ferramenta no processo e, ao mesmo tempo, uma integração. O conceito integração no ADS é de grande importância e de propósito fundamental. Sua importância atual é de tamanha prioridade que é a integração que define todas as regras e diretrizes que governarão o uso das ferramentas e combinação entre elas numa harmonia em todas as etapas do processo, produzindo assim um maior suporte efetivo destas ferramentas ao longo de todo o desenvolvimento do software.

Assim, com a integração, as ferramentas se tornam fator de importância dentro de um ADS. Porém, é preciso distinguir os vários níveis que existem desta integração de ferramentas num ADS. Vejam eles: Integração de Dados (habilidade de compartilhar informação do projeto); Integração de Apresentação (torna as interfaces do ADS consistentes); Integração de Controle (gerenciamento do processo); Integração de Conhecimento (conhecimento das informações de natureza semântica); Integração de Processo (as ferramentas e os recursos existentes em um processo se relacionam).

> Para que as dimensões possam ser trabalhadas em ADSs, é comum o uso de diversas tecnologias de apoio. Dessa forma, para integrar dados, pode-se utilizar, por exemplo, um sistema gerenciador de banco de dados ou arquivos XML; a integração de controle pode usar orientação a objetos, orientação a aspectos, uma estrutura Modelo-Visão-Controle ou agentes; linguagens de modelagem de processos são úteis à integração de processos; e a integração de conhecimento utiliza-se de sistemas baseados em conhecimento, técnicas de apoio à gerência de conhecimento, ontologias, agentes e máquinas de inferências. (RUY, 2006, f. 22).

A multiplicidade de modelos e normas nas ferramentas podem ser um problema no desenvolvimento do software. Problemas de integração irão surgir e comprometer a qualidade dos processos de software. E como melhorar a comunicação entre ferramentas e desenvolvedores em um ADS? O crescimento rápido, contínuo e múltiplo de informações torna cada vez mais complexo o acesso integrado e efetivo destas informações. Gasta-se tempo e recursos para pouca efetivação das informações espalhadas em diversas ferramentas. Neste caso, as chamadas "ontologias" surgem como solução para este problema. "Ontologias têm se tornado populares, em grande parte, pelo fato de terem como objetivo promover um entendimento comum e compartilhado sobre um domínio, que pode ser comunicado entre pessoas e sistemas de aplicação", especifica FALBO *et al*[1].

O conceito de ontologia é claro e objetivo: tenta resolver problemas de comunicação dentro das organizações, o que pode causar dificuldades no apontamento dos requisitos de um sistema. As ontologias também definem um vocabulário comum que facilita a comunicação, integração, armazenamento e representação do conhecimento.

Podem ser usadas como uma estrutura unificadora para dar semântica e uma representação comum à informação. Também, podem ser úteis na construção de ADSs e contribuem muito para a integração, pois melhoram a comunicação entre

[1] FALBO *et al*, 2004, fl. 2.

desenvolvedores e ferramentas em um ADS. A utilização de ontologias, com isso, podem trazer muitos benefícios. “A principal contribuição das ontologias para ADSs é prover um conjunto de conceitos e restrições bem definidos, determinando um vocabulário comum que pode ser compartilhado por pessoas e ferramentas. A premissa de ODE é que, se as ferramentas são construídas baseadas em ontologias, a sua integração é facilitada”, diz BERTOLLO[2].

2.1.1. Tipos de Ambientes de Desenvolvimento de Software e importâncias.

Os tipos de ADS nascem ao longo da história a partir das aplicações que darão mais ênfase em alguma das dimensões apresentadas. A seguir são apresentados três tipos de ADS, cada um com sua ênfase e especificidade trabalhando com diversas ferramentas para seu melhor desempenho.

2.1.1.1. Ambientes de Desenvolvimento de Softwares Centrados em Processos (ADSCP).

Um produto de software precisa ser desenvolvido num ADS com ferramentas tecnicamente especializadas e, mais do que isso, utilizando recursos de um processo de software, constituído de um conjunto de atividades e recursos utilizados em produtos. As ADSCPs têm como objetivo principal oferecer apoio e ferramentas para diversos tipos de processos de software, determinando assim como o ambiente deve se comportar neste momento. As ADSCPs irão integrar diversas ferramentas dando suporte técnico ao desenvolvimento do produto de software e apoiando sua modelagem juntamente com a execução do processo de software escolhido.

> Os Ambientes Centrados em Processo (ACPs): tais ambientes concentram-se em como computadores podem ser utilizados no desenvolvimento de sistemas, focalizando a integração de controle como mecanismo para permitir a coordenação de equipes, o gerenciamento de configurações e o gerenciamento de contexto, ou seja, o controle do que é visível para cada ferramenta e para cada usuário, simplificando as interações do usuário com as ferramentas e outros usuários. Porém, para que todos estes benefícios possam

[2] BERTOLLO, 2006. fl. 12.

> ser alcançados, um ACP precisa ter seus processos modelos para que possam ser automatizados[3].

Os processos de software constituem-se de vários elementos/ferramentas, como: atividade; artefatos; recursos; processo; projeto. Estas fases serão vistas com mais detalhes no tópico sobre Processos de Software deste capítulo.

I - Ambientes ODE (ADS baseados em Ontologias).

Uma característica objetiva das ontologias é resolver problemas de comunicação dentro das organizações, especificamente no que diz respeito à identificação dos requisitos de um sistema. As ontologias também contribuem dentro do conceito de comunicação, para o compartilhamento de um vocabulário mais definido, no qual a uniformidade deste vocabulário evitará ambigüidades e inconsistências no trabalho de um ADS. Com isso, facilita o entendimento compartilhado e a comunicação entre pessoas com diferentes necessidades e pontos de vista. Várias são as ontologias que compõem a base ontológica de ODE, dentre elas tem-se as ontologias de processo de software, de qualidade de software, de artefatos de software e de riscos de software. Ontologias estas que são usadas, dentre outros, para estruturar o ambiente e sua infra-estrutura de gerência de conhecimento e para estabelecer uma forma padrão de comunicação entre os agentes que atuam no ambiente, conforme FALBO *et al*[4].

Por fim, "ODE (Ontology-based software Development Environment) é um ADS Centrado em Processo, que realiza Gerência de Conhecimento em Egenharia de Software, fundamentando-se especialmente em sua base ontológica"[5].

II - Estação TABA.

A Estação TABA, conforme sua primeira definição (ROCHA *et al.*, 1990), é um meta-ambiente capaz de gerar, através de instanciação, ambientes de desenvolvimento de software adequados às particularidades de organizações, processos

[3] MIAN *et a*, 2001. fl. 3.
[4] FALBO *et al.* fl. 3.
[5] RUY, 2006, p. 22.

de desenvolvimento e de projetos específicos. ROCHA *et al.* (1990) definem meta-ambiente como um ambiente que abriga um conjunto de programas que interagem com os usuários para definir interfaces, selecionar ferramentas e estabelecer os tipos de objetos que irão compor o ambiente de desenvolvimento específico. Um objetivo claro da Estação TABA é o auxílio na definição, implementação e execução de ADS adequados a contexto específicos, os ditos Ambientes de Desenvolvimento de Software Orientados a Organização (ADSOrg).

A Estação TABA também apresenta a característica de um ADS Centrado em Processo, no qual este trabalho está focando, pois possibilita modelar explicitamente um processo de software e controlá-lo no ambiente instanciado. O ambiente possibilita, ainda, a gestão/administração do conhecimento da organização, definindo um tipo de banco de dados de conhecimento que pode ser acessado pelos ADSOrg instanciados.

2.1.1.2. Ambientes de Desenvolvimento de Softwares Orientados a Domínio (ADSOD).

No ADSOD só é identificado quando o ambiente passa a considerar o conhecimento do domínio da aplicação, para prover apoio de gerência de conhecimento. Este ADS tem sua importância porque a motivação para esta evolução foi a identificação de que um dos principais problemas no desenvolvimento de software era o desconhecimento do domínio por parte dos desenvolvedores de software. A solução dos especialistas foi a criação do ADSOD, constatando que o melhor desenvolvimento e a manutenção de um produto de software quando capazes de fornecer conhecimento do domínio aos desenvolvedores.

Este tipo de ambiente propõe um apoio ao entendimento do domínio para os desenvolvedores, principalmente aqueles que não têm familiaridade ou experiência em realizar trabalhos no domínio considerado. ADSODs são definidos tendo como base os tradicionais ADSs, mas incorporando um novo fator: o conhecimento de um domínio específico (OLIVEIRA *et al.*, 2004).

Assim, podem ser citadas duas características específicas e essenciais para seu entendimento e utilização, como: o conhecimento do domínio deve ser capturado, modelado e armazenado para uso no ambiente (aqui entra o uso de ontologias, no qual o conhecimento de domínio pode ser definido sobre uma base conceitual robusta, que facilita a sua

manipulação) e o ambiente deve suportar a disseminação e uso do conhecimento do domínio (aqui entra o conceito de gerência de conhecimento, pois o conhecimento de domínio pode ser gerenciado).

2.1.1.3. Ambientes de Desenvolvimento de Softwares Orientados à Organização (ADSOrg).

Este tipo de ADS tem sua origem na própria necessidade da organização gerenciar o seu próprio conhecimento e o seu próprio domínio. "Um ADSOrg apóia a atividade de Engenharia de Software em uma organização, fornecendo conhecimento acumulado pela organização e relevante para esta atividade, ao mesmo tempo em que apóia, a partir dos projetos específicos, o aprendizado organizacional em Engenharia de Software"[6].

Um ADSOrg necessita dos seguintes requisitos: ter uma representação da estrutura organizacional; reter conhecimento especializado sobre desenvolvimento e manutenção de software; permitir a utilização deste conhecimento em projetos; apoiar a atualização constante do conhecimento armazenado no ambiente; facilitar a localização de especialistas de organização.

Um ADSOrg evita a dispersão ao longo da estrutura organizacional do conhecimento de software e, conseqüentemente, sua dificuldade de acesso e perdas. Um ADSOrg transforma em conhecimento dos gerentes de projeto e em atividades centradas no conhecimento as atividades iniciais dos projetos de desenvolvimento do software.

> O ADSOrg tem como objetivo: (a) apoiar os desenvolvedores de software na execução de suas atividades, fornecendo todo o conhecimento que tenha sido capturado e acumulado pela organização por sua importância para o desenvolvimento e a manutenção de software, e (b) apoiar o aprendizado organizacional em Engenharia de Software a partir do aprendizado dos desenvolvedores da organização nos projetos de software específicos. A finalidade é evitar erros já cometidos e possibilitar a reutilização de soluções já aprovadas na execução de tarefas similares, buscando melhorar a produtividade e a qualidade, bem como diminuir custos. (ANDRADE, 2005, fl. 43).

[6] RUY, 2006, fl. 27.

O ADSOrg também é um ambiente instanciado a partir de um Ambiente Configurado. Com isso, um ambiente configurado necessariamente possui uma ferramenta de apoio à instanciação de processos, no qual tem a capacidade de gerar ambientes para projetos específicos. E contém, assim, outras ferramentas de apoio ao desenvolvimento.

2.2. Processos de Desenvolvimento de Software.

Basicamente podemos definir um processo como um conjunto de passos parcialmente ordenados, constituídos por atividades, métodos, práticas e transformações, usado para atingir uma meta. Esta meta geralmente está associada a um ou mais resultados concretos finais, que são os produtos da execução do processo (PAULA FILHO, 2003, p. 11). Já num PDS entra outros detalhes e conceitos (conjunto de atividades que leva à produção de um produto de software): desenvolvimento, manutenção, aquisição e contratação de software. Porém, uma escolha é necessária e prioritária: a escolha de um modelo de ciclo de vida (especificação – produto). Existem vários modelos e fases como será visto adiante.

Embora existam muitos processos de software, podem ser especificadas algumas atividades fundamentais que são comuns a todos eles, como[7]: Especificação de software; Projeto e implementação de software; Validação de software; Evolução de software.

2.2.1. Modelos, Fases e Processos.

I - Modelos.

Dentre os modelos que existem podemos enumerar os seguintes com seus subprocessos[8]: O modelo de ciclo de vida em Cascata: requisitos; análise; desenho; implementação; testes. O modelo de ciclo de vida em cascata com realimentação: requisitos; análise; desenho; implementação; testes (com a diferença da realimentação

[7] cf. SOMMERVILLE, 2007, p. 43.
[8] cf. PAULA FILHO, 2003, p. 13-15.

em cada subprocesso do ciclo em Cascata). O modelo de ciclo de vida em Espiral: desenvolvimento; avaliação; análise dos ricos; ativação; planejamento da próxima interação. O modelo de ciclo de vida de Entrega Evolutiva: requisitos; análise; desenho arquitetônico; desenho detalhado; implementação; testes; avaliação da iteração.

> O desenvolvedor deve definir ou selecionar um modelo de ciclo de vida apropriado para o contexto, a magnitude e a complexidade do projeto a ser desenvolvido. Além disso, deve selecionar e adequar as normas, os métodos, os procedimentos, as linguagens de programação e as ferramentas utilizadas no projeto. Os métodos de desenvolvimento seguem um paradigma que determina o tipo de desenvolvimento a ser seguido e as ferramentas a serem utilizadas em todo o ciclo de vida (ROCHA *et al,* 2001, p. 44).

II - Fases.

As fases de um processo de software irão modificar conforme cada linha de desenvolvedores, porém algumas fases são primordiais para todo desenvolvedor e processo de desenvolvimento de um software. Abaixo temos objetivamente a descrição das fases necessárias para um desenvolvimento de um software[9]:

- Implementação de um processo: aqui o desenvolvedor irá escolher um modelo de ciclo de vida, dentro do contexto do software. E também adequar as normas, os métodos, os procedimentos, as linguagens de programação e as ferramentas a serem utilizadas no processo.
- Análise de requisitos do sistema: nesta fase são especificados os requisitos (uma série de sentenças) significativos que irão compor o sistema a ser desenvolvido.
- Projeto arquitetural do sistema: é a fase que entra a arquitetura contendo os elementos de hardware, software e os procedimentos manuais. E conseqüentemente a comunicação entre estes elementos.
- Análise dos requisitos do software: refere-se a especificação dos requisitos de software no qual especifica a análise dos itens para determinar os requisitos funcionais e não-funcionais, como: segurança, usabilidade, documentação, instalação e aceitação, operação, manutenção, etc.
- Projeto arquitetural do software: fase que determina a arquitetura de um sistema de software definindo suas estruturas gerais descrevendo assim os elementos que compõem os sistemas e as interações entre eles.

- Projeto detalhado do software: nesta fase o detalhe do projeto de software engloba o nível de interfaces, classes ou componentes preexistentes.
- Codificação e teste do software: o desenvolvedor deve codificar e documentar cada unidade de software e definir a base de dados usando técnicas de implementação que produzam um código eficiente e livre de erros (ROCHA, 2001, p. 46).
- Integração do software: a integração dos componentes de software, nesta fase, exige que o desenvolvedor defina um plano. Faz-se aqui um teste de integração.
- Teste de qualificação de software: os testes feitos nesta fase são os seguintes: testes de cobertura, atendimento aos resultados esperados e viabilidade de integração e de operação. Inclui-se aqui a verificação e a validação dos requisitos previamente estabelecidos.
- Instalação do software: estabelece um plano de instalação do software no ambiente previsto e sua documentação, auxiliando o usuário que será o dono do software.
- Apoio à aceitação do software: auxílio ao usuário adquirente do software para a aceitação do mesmo e documentando esta fase.

III – Processos (Exemplos)[10].

O uso de processos definidos tem como característica primeira de servir como modelo para o desenvolvimento direto de softwares, estando eles abertos para a maturidade. São modelos de forma prática, precisa e quantitativa. Existem vários tipos de exemplos de processos, no qual serão citados os seguintes:

O Processo Pessoal de Software (PSP).

Este tipo de processo (HUMPHREY, 1990) possui uma série de exemplos (processos pessoais) que podem ser aprendidos em uma disciplina de engenharia de software. Constituem um conjunto de formulários, scripts e relatórios predefinidos,

[9] cf. ROCHA, 2001, p. 43-47.
[10] cf. PAULA FILHO, 2003, p. 16-52.

sendo todos processos individuais. Os estágios constituídos são os seguintes: Processos pessoais básicos; Processos pessoais com planejamento; Processos pessoais com gestão de qualidade; Processos pessoais cíclicos.

O Processo de Software para Equipes (TSP).

É autor deste processo HUMPHREY (HUMPHREY, 1990). Uma das características do TSP é utilizar o modelo em espiral. As fases do TSP são as seguintes: Lançamento; Estratégia; Planejamento; Requisitos; Desenho; Implementação; Testes; Post-mortem. Os participantes do time de desenvolvedores são organizados de tal forma que cada desenvolvedor desempenhe um ou dois papéis gerenciais bem-definidos, além de dividir a carga de desenvolvimento. O TSP enfatiza algumas áreas que correspondem às áreas do nível 2 do CMM (PAULA FILHO, 2003, p. 18).

O Processo Unificado (UP).

Este exemplo de processo utiliza a UML (sendo autores: BOOCH, Jacobson e Rumbaugh, notação de modelagem orientada em objetos) como notação de uma série de modelos que compõem os principais resultados das atividades do processo. O UP tem as seguintes características centrais: é dirigido por casos de uso; é centrado na arquitetura; é iterativo e incremental. Os fluxos de trabalho (*workflows*) são subprocessos que fazem parte das atividades técnicas do UP. As fases do UP são estas: Concepção; Elaboração; Construção; Transição; Requisitos; Análise; Desenho; Implementação; Testes.

Um exemplo de processo derivado do UP é o *Rational Unified Process* (RUP)[11]. É contido de vários elementos dos modelos genéricos de processo. O RUP pode ser descrito pelas três perspectivas: uma perspectiva dinâmica; uma perspectiva estática; uma perspectiva prática.

São quatro as fases do RUP: Concepção; Elaboração; Construção; Transição. Já os *workflows* estáticos no RUP são estes: Modelagem de negócios;

[11] cf. SOMMERVILLE, 2007, p. 54-56.

Requisitos; Análise e projeto; Implementação; Teste; Implantação; Gerenciamento de configuração e mudanças; Gerenciamento de projetos; Ambiente.

Também no RUP pode-se ver boas práticas de engenharia de software, são recomendadas seis melhores práticas fundamentais: Desenvolver o software iterativamente; Gerenciar requisitos; Usar arquiteturas baseadas em componentes; Modelar o software visualmente; Verificar a qualidade do software; Controlar as mudanças do software.

O RUP contribui favoravelmente para a qualidade de um software, no qual pode ser medida nos seguintes fatores[12]: Corretude; Eficiência; Performance; Confiabilidade/robustez; Escalabilidade em ambiente multi-usuário; Segurança adequada; Facilidade de manutenção; Facilidade de utilização; Portabilidade.

O Processo Práxis (PP).

Como o próprio nome diz, o Processo Práxis é um processo prático desenhado para dar suporte a projetos didáticos, no qual dá ênfase no desenvolvimento de aplicativos gráficos interativos, sempre baseados na tecnologia orientada a objetos.

O PP abrange as seguintes fases: Concepção; Elaboração; Construção; Transição. E os seguintes fluxos técnicos e gerenciais: Fluxos Técnicos → Requisitos; Análise; Desenho; Implementação; Testes; Engenharia de Sistemas. Fluxos Gerenciais → Gestão de projetos; Gestão de qualidade; Engenharia de processos. No PP também é descrito os subfluxos gerenciais.

Processos Agéis.

O surgimento dos processos agéis é em detrimento, ou como alternativa, de uma evolução em relação aos processos tradicionais. Estes tipos de processos possuem as seguintes características: contribui na desburocratização no processo de desenvolvimento de software; criar uma adaptação e estabelece fortalecimento com as mudanças suscetíveis; possuem funcionalidades que agregam maior valor ao negócio

[12] cf. QUADROS, 2002, p. 24-26.

ternando-as priorizadas; versões de software com um executável em curto espaço de tempo; não criar documentação em excesso sem necessidade; a documentação especifica apenas as descrições relatadas pelos usuários e as descrições do funcionamento do sistema; estes tipos de processos possuem o foco principal nas pessoas (usuários) e não somente nos processos. Eis nomes de alguns processos agéis: SCRUM; Extreme Programming (XP); AgileUP; Feature-Driven Developmente (FDD); DSDM; OpenUP; e outros.

2.3. UML.

A UML (*Unified Modeling Language*) pode ser definida como uma linguagem-padrão para a construção/elaboração de estrutura de projetos de softwares, sendo assim uma linguagem visual utilizada para a modelagem de sistemas computacionais orientados a objetos. É uma linguagem para modelagem de sistemas, utilizando diagramas gráficos que ajudam a perceber, visualizar, especificar, construir e a documentar os artefatos de softwares.

> A UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir sistemas de informação corporativos a serem distribuídos a aplicações baseadas em Web e até sistemas complexos embutidos de tempo real. É uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação desses sistemas. Apesar de sua expressividade não é difícil compreender e usar a UML. Aprender a aplicar a UML de maneira efetiva tem início com a formação de um modelo conceitual da linguagem, o que pressupõe o entendimento de três principais elementos: os blocos básicos de construção da UML, as regras que determinam como esses blocos de construção deverão ser combinados e alguns mecanismos básicos que se aplicam a toda a linguagem (BOOCH *et al*, 2005, p. 13).

A UML oferece uma linguagem para a modelagem das atividades de planejamento do projeto e de gerenciamento de versões, utilizada assim para a construção de vários artefatos na documentação de um software. Seus modelos podem estar diretamente conectados às várias linguagens de programação, mapeando seus modelos em linguagens como: Java, C++, Visual Basic ou até tabelas de banco de dados orientados a objetos. Permitindo assim, neste mapeamento, uma engenharia de produção e geração de código.

A UML não é apenas utilizada para modelagem de software, mas também em outras áreas que explicitam melhor o fluxo de trabalho em um sistema. Eis alguns

exemplos de outras aplicações da UML: Eletrônica médica; Científicos; Defesa de Espaço aéreo; Vendas de varejo; Serviços bancários e financeiros; Sistemas de informações corporativos; Telecomunicações; Transportes; Indústrias diversas; etc.

Na UML podemos identificar quatro tipos de itens: Itens estruturais (classes; interfaces; colaborações; casos de uso; classes ativas; componentes; artefatos; nós); Itens comportamentais (interação; máquina de estado; atividade); Itens de agrupamentos (pacotes); Itens anotacionais (notas).

Nos relacionamentos da UML podemos citar quatro tipos: dependência; associação; generalização; realização. Existindo também suas variações, como: refinamentos, rastros, inclusões e extensões.

Um item importante é a apresentação gráfica que permite visualizar o sistema de forma estática e dinâmica através de diferentes perspectivas, em diagramas UML.

> Um diagrama é a apresentação gráfica de um conjunto de elementos, geralmente representadas como gráficos de vértices (itens) e arcos (relacionamentos). São desenhados para permitir a visualização de um sistema sob diferentes perspectivas; nesse sentido, um diagrama constitui uma projeção de um determinado sistema. Em todos os sistemas, com exceção dos mais triviais, um diagrama representa uma visão parcial dos elementos que compõem o sistema (BOOCH *et al*, 2005, p. 26).

A UML constitui vários diagramas para esta melhor visualização do sistema, dentre eles estão: Diagrama de Classes; Diagrama de Objetos; Diagrama de Componentes; Diagrama de Estruturas Compostas; Diagrama de Casos de Uso; Diagramas de Seqüências; Diagrama de Comunicações; Diagrama de Gráficos de Estados; Diagrama de Atividades; Diagrama de Implantação; Diagrama de Pacote; Diagrama de Temporização; Diagrama de Visão Geral da Interação. A importância destes diagramas se dá conforme a óptica que o diagrama analisará o sistema. É como se o sistema fosse modelado em camadas, sendo que alguns diagramas enfocam o sistema de forma mais geral, apresentando uma visão externa do sistema. A utilização de diversos diagramas permite que falhas sejam descobertas, diminuindo a possibilidade da ocorrência de erros futuros (GUEDES, 2008, p. 27).

Como a UML é uma linguagem inserida no Paradigma de Orientação a Objetos, pode-se levar em consideração vários conceitos que acompanham este tipo de paradigma, como: Classificação; Abstração; Instanciação; Classe de Objetos; Atributos

ou Propriedades; Métodos ou Comportamentos; Visibilidade; Herança; Herança Múltipla; Polimorfismo.

A UML 2.0 foi criada com o objetivo de abordar uma metamodelagem que adapta técnicas de especificação formal. Ela oferece as vantagens de ser mais intuitiva e pragmática (GUEDES, 2008, p. 264). Nesta nova versão alguns princípios são especificados, como: Modularidade; Divisão por camadas (Layering); Particionamento; Extensibilidade; Reuso. Assim, alguns diagramas tiveram sua nomenclatura modificada e acréscimo de alguns novos, criados pela nova versão. São eles: Diagrama de Máquina de Estados (Diagrama de Gráfico de Estados, da versão anterior); Diagrama de Comunicação (Diagrama de Colaboração, da versão anterior); Diagrama de Estrutura Composta; diagrama de Interação Geral; Diagrama de Tempo.

2.4. Arquitetura Dirigida pelo Modelo (MDA).

A MDA (Model Driven Architecture, Arquitetura Dirigida pelo Modelo) pode ser entendida como uma metodologia de desenvolvimento de software criada pela OMG (Object Management Group. Sua característica consiste na integração da modelagem no processo de desenvolvimento do software. Esta metodologia define que o processo de desenvolvimento de software deve estar direcionado pela atividade de modelagem do sistema, no nível conceitual, sem qualquer dependência com plataforma ou implementação. Os tipos de modelos criados são mais formais, com quase nenhuma ambigüidade. MDA compreende três etapas principais: construção de um modelo com alto nível de abstração, independente de qualquer tecnologia; criação de Modelos Específicos de Plataforma (PSM); e transformar um PSM em código. Algumas vantagens: produtividade; portabilidade; interoperabilidade; e outros.

A figura abaixo mostra os principais artefatos da MDA:

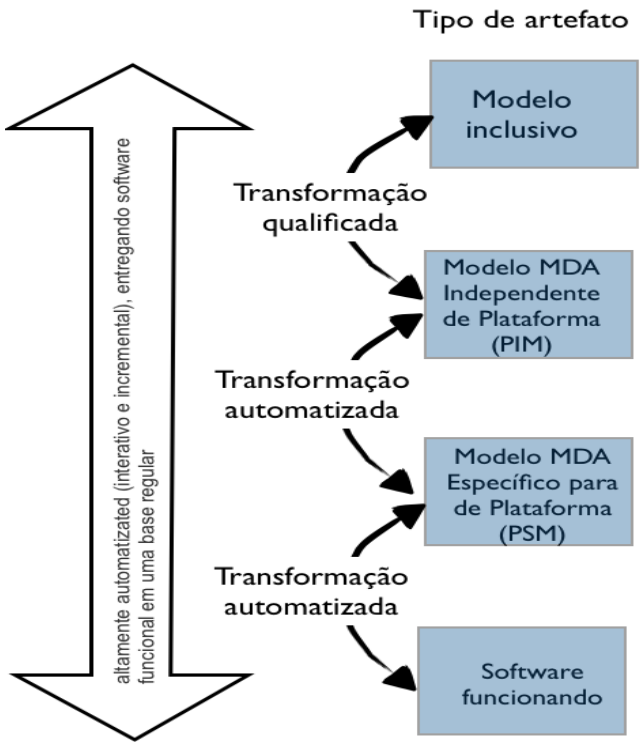


Figura 1 – Artefatos da MDA.

FONTE: http://www.agilemodeling.com/essays/agileMDA.htm

3 FERRAMENTAS CASE.

Nesta parte do trabalho será tratado em continuidade com a primeira parte a integração de ferramentas CASE com os ADSs. Foi visto a abrangência de um ADS, no qual disponibiliza várias ferramentas de apoio ao longo de todo o processo de desenvolvimento de um software. Porém, agora, será visto mais especificamente a utilização de uma ferramenta CASE (implicações), que se destina a apoiar apenas uma etapa do processo de desenvolvimento de software.

Os conceitos de Ferramentas CASE e ADS (a integração das duas) vão contribuir de maneira significativa para o engenheiro de software que está sempre em busca da combinação de melhores técnicas, métodos e ferramentas.

Atualmente, as ferramentas CASE são amplamente utilizadas pela indústria de software. Ferramentas comerciais apóiam a realização de grande parte das atividades que compõem o processo de software, principalmente as relacionadas à construção. Entretanto, cada uma dessas ferramentas geralmente atende a uma, ou a poucas atividades, resolvendo apenas problemas pontuais. Isso implica na utilização de várias ferramentas para que se possa atender às diversas atividades do processo. Além disso, são raras as vezes em que conseguem comunicar-se entre si, trocando informações ou serviços (RUY, 2006, f. 20).

3.1. O Conceito de Ferramentas CASE: principais características e importâncias.

Um conceito simples de CASE pode ser definido assim: “CASE é a automação do desenvolvimento de software”[13]. Contudo pode-se também definir CASE como um conjunto de técnicas e ferramentas informatizadas que auxiliam o engenheiro de software no desenvolvimento de aplicações (softwares, sistemas), com o objetivo de diminuir o respectivo esforço e complexidade (maior qualidade e melhor tempo), de melhorar o controle do projeto, de aplicar sistematicamente um processo uniformizado e de automatizar algumas atividades, nomeadamente a verificação da consistência e qualidade do produto final e a geração de artefatos. Uma ferramenta CASE é

[13] MCCLURE, Carma. *CASE is software automation***.** New Jersey : Prentice_hall, 1989.

simplesmente um produto informatizado destinado a suportar uma ou mais atividades (ferramentas) de engenharia de software, relacionadas com uma (ou mais) metodologia(s) de desenvolvimento[14]. As ferramentas CASE possuem a analogia de serem uma caixa de ferramentas para o engenheiro de software, proporcionando assim ao mesmo tempo uma potencialidade de automatizar as atividades que antes eram feitas manualmente, facilitando assim as informações da engenharia. Possuem um potencial grande no avanço tecnológico em relação ao desenvolvimento de ferramentas.

> A Engenharia de Software Auxiliada por Computador (CASE – *Computer-Aided Software Engineering*) é o nome dado ao software usado para apoiar as atividades de processo de software, como engenharia de requisitos, projeto, desenvolvimento de programas e teste. As ferramentas CASE, portanto, incluem editores de diagramas, dicionário de dados, compiladores, debuggers, ferramentas de construção de sistemas etc. A tecnologia CASE fornece apoio ao processo de software pela automação de algumas atividades de processo e pelo fornecimento de informações sobre o software que está sendo desenvolvido (SOMMERVILLE, 2007, p. 56).

Especificamente, as ferramentas CASE estão mais concentradas na arquitetura do sistema[15]. A presença de ferramentas CASE é vital hoje em dia para o bom funcionamento de uma empresa desenvolvedora de software. Elas existem auxiliando todo o ciclo de desenvolvimento (análise, projeto, implementação e teste) e são também de suma importância para a manutenção do software. Há também ferramentas CASE para apoiar a gerência dos projetos de desenvolvimento[16]. As ferramentas CASE, na sua originalidade, devem executar as seguintes tarefas[17]: fracionamento da complexidade; adequação a um público diversificado; mais baratas que a construção em si; quantitativas e verificáveis; de fácil manutenção; orientação gráfica.

São varias as características que dão importância às ferramentas CASE, dentre muitas podemos especificar estas: focaliza quase que exclusivamente a solução de problemas[18]; geração automática de programas (código) em nível de uma especificação em projeto; padronização dos produtos; automação do Processo de Software; reengenharia, engenharia reversa e suporte à reestruturação; interoperabilidade entre ferramentas; manutenção de coleções de dados e comunicação;

---

[14] SILVA *et al*, 2011, p. 397.
[15] HOFFMANN, *Avaliação da qualidade da ferramenta CASE System Architect baseada na norma ISO/IEC 14102*. 2001, f. 12.
[16] *Idem*, f 16.
[17] *Idem*, f. 18-19.

tornar o trabalho /desenvolvimento menos tedioso; impor padrões de notações e métodos entre os usuários; facilitar a verificação de consistência e completude do projeto, documentar o código dos sistemas; aumentar a produtividade e reduzir os custos de desenvolvimento; ajudar a melhorara a documentação e manutenção; possibilitar que problemas no desenvolvimento seja descobertos mais cedo possível evitando a propagação entre as diversas fases; ajudar a melhorar a qualidade (confiabilidade; reusabilidade; etc).

As etapas[19] para implantação de uma Ferramenta CASE são as seguintes: Estabelecer padrões; Treinamento do pessoal; Projeto piloto; Formar equipe de suporte ao CASE; Estabelecer mecanismos para guiar o uso do CASE.

3.2. Funcionalidades (Aplicações) das Ferramentas CASE.

O uso de uma ferramenta CASE pode conter as diversas atividades, como: desenvolvimento dos modelos gráficos de sistema; o uso de um dicionário de dados, na compreensão de um projeto; geração de interfaces interativas; o debugging do programa; tradução automática de programas, fazendo a integração entre uma linguagem antiga passando para uma nova linguagem.

> O maior benefício do uso de ferramentas CASE no processo de software pode advir com a integração. Os benefícios de I-CASE *(Integrated CASE)* incluem: (1) eficiência na transferência de informações entre as ferramentas; (2) redução do esforço em atividades como gerenciamento de configuração, garantia de qualidade, produção de documentos; (3) aumento do controle do projeto e (4) melhoria na cooperação entre membros de um projeto. Naturalmente alguns desafios surgem com I-CASE: demanda de consistência na representação da informação, interfaces padronizadas entre as ferramentas, mecanismos homogêneos para comunicação, abordagem efetiva que torne I-CASE compatível com várias plataformas de hardware e sistemas operacionais (FARIAS, f. 17-18).

Dentre as vantagens de uma Ferramenta CASE podemos elencar as seguintes[20] (conforme outros autores[21] também)[22]:

---

[18] PRESSMAN, 1995, p. 946.
[19] *Ferramentas CASE*, profª Vera Werneck.
[20] SILVA *et al*, 2011, p. 409.
[21] FISCHER, 1990.
[22] CHIKOFSKY, Elliot. *Computer-Aided Software Engineering (CASE)*. COMPUTER IEEE Computer Society, 1993.

- Uniformização do processo de desenvolvimento, das actividades realizadas, e dos artefatos produzidos.
- Reutilização de vários artefatos ao longo do mesmo projeto, e entre projetos, promovendo o consequente aumento da produtividade.
- Automatização de actividades, com particular destaque ao nível da geração de código e de documentação.
- Diminuição do tempo de desenvolvimento, recorrendo à geração automática de diversos artefatos do projeto, ou à realização de outros previamente existentes.
- Integração de artefatos produzidos em diferentes fases do ciclo de desenvolvimento de software, em que os outputs de uma ferramenta são utilizados como inputs de outra.
- Demonstração da consistência entre os diversos modelos e possibilidade de verificar a correcção do software.
- Qualidade do produto final superior.
- E outras como: Especificações completas dos requisitos; Especificações minuciosas do projeto; Especificações atuais do projeto; Código altamente flexível e de fácil manutenção.

Como também podemos citar algumas desvantagens: incompatibilidade de ferramentas; elevado custo de ferramentas como preparação do pessoal para utilização destas ferramentas; elevada curva de aprendizagem; limitações na flexibilidade da documentação. A estratégia de introdução das ferramentas CASE numa organização pode ser diversa, nomeadamente: Suite **:** selecção de um conjunto integrado de ferramentas, todas do mesmo fornecedor. Best-of-breed **:** selecção das melhores ferramentas para cada funcionalidade, suportadas por um repositório integrado. Pontual **:** selecção de ferramentas para cobrir áreas pontuais.

3.3. Arquitetura de uma Ferramenta CASE.

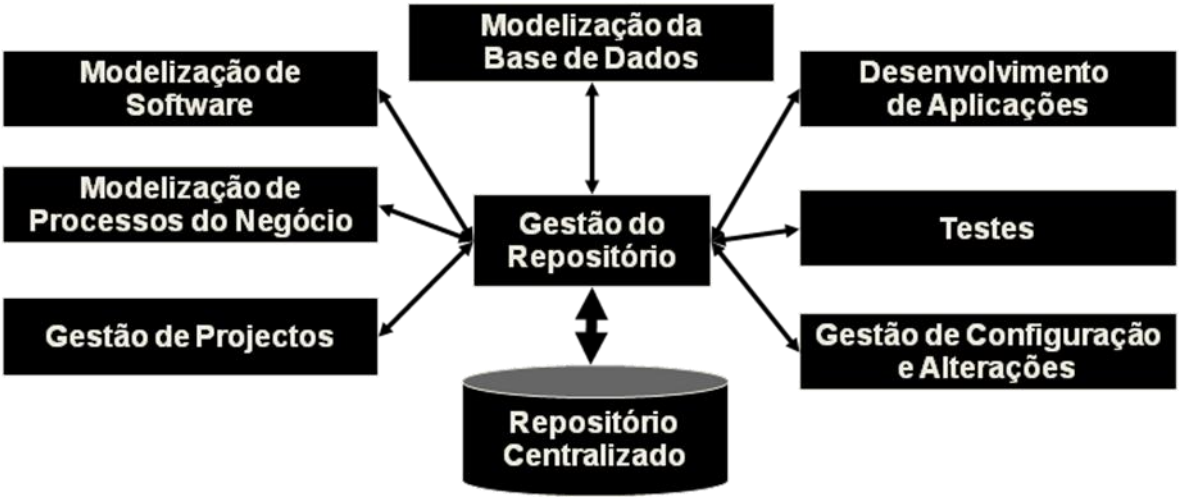
A maioria das ferramentas CASE especializa-se sobretudo numa tarefa específica do processo de desenvolvimento de software. Algumas concentram-se na disponibilização de funcionalidades relevantes para a fase de concepção (por exemplo,

.

elaboração de diversos diagramas), enquanto outras estão particularmente direccionadas para a fase de implementação (por exemplo, desenvolvimento visual, geração de código e apoio à realização de testes).

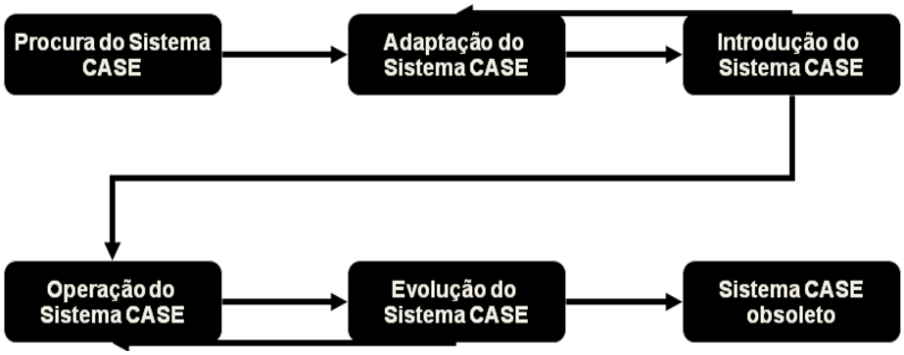
A arquitetura global CASE deve ser flexível o suficiente para acomodar a metodologia e os padrões e práticas de administração de projeto usados atualmente pela organização, em vez de forçar a organização a aceitar as restrições arbitrárias impostas pelo próprio produto. Idealmente, o conjunto de ferramentas deve adaptar-se às necessidades da empresa e não o contrário.

A arquitetura típica das ferramentas CASE é constituída por um conjunto de aplicações/componentes, suportados por um repositório integrado, como se representa na seguinte figura[23]:



Fonte: www.estig.ipbeja.pt/~eides/**CASE**_4178.ppt

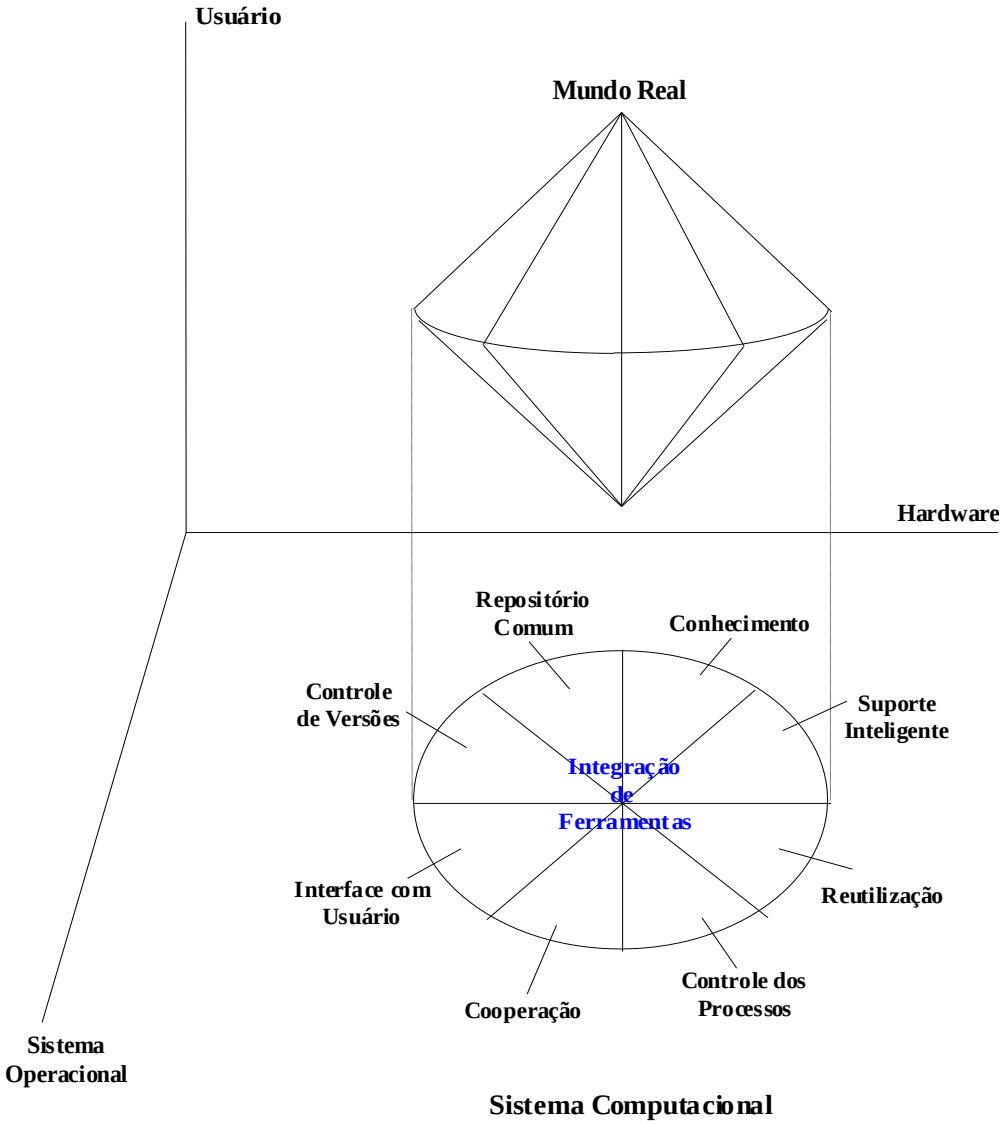
Já o ciclo de vida de uma ferramenta CASE pode considerado conforme a figura abaixo[24]:



Fonte: www.estig.ipbeja.pt/~eides/**CASE**_4178.ppt

[23] *Idem*, p. 403.
[24] *Idem.*

Arquitetura de Integração:



Fonte: www.lbd.dcc.ufmg.br/colecoes/sbes/2004/011.pdf

3.4 Tipos (Classificação) de Ferramentas CASE.

Dentre as diversas formas de classificar ferramentas CASE está especificado três perspectivas neste trabalho que segue uma delas[25]: perspectiva funcional; perspectiva de processo; e perspectiva de integração. Podendo ser classificadas em três categorias: ferramentas; workbenches e ambientes (alguns autores traz estas categorias:

[25] SOMMERVILLE, Ian. *Engenharia de Software*. 8. ed. São Paulo: Pearson Addison-Wesley, 2007, p. 57.

Lower CASE - ferramentas de codificação (front-end); Upper CASE - ferramentas de análise, projeto e implementação; Integrated CASE - união de Upper e Lower CASE). Ou também, divididas em três níveis[26]: tecnologia de suporte ao processo de produção; tecnologia de gerenciamento de processo; e tecnologia Meta-CASE.

Os tipos de ferramentas podem ser assim classificados, com base na funcionalidade[27]: ferramentas de planejamento; ferramentas de edição; ferramentas de gerenciamento de mudanças; ferramentas de gerenciamento de configuração; ferramentas de prototipação; ferramentas de apoio a métodos; ferramentas de apoio ao reuso; ferramentas de processamento de linguagens; ferramentas de análise de programa; ferramentas de teste; ferramentas de depuração; ferramentas de documentação; ferramentas de reengenharia. Porém, cada um destes tipos de ferramentas é útil (contribui) dentro de cada atividade que faz parte em um processo de desenvolvimento de software, como: especificação, projeto e implementação, validação e evolução de software.

Há a classificação[28] com base nas fases do PDS, como: ferramentas de geração de dados de testes; ferramentas de modelagem e simulação; ferramentas de transformação de programas; ferramentas de depuração interativa; ferramentas de análise de programas; ferramentas de processamento de linguagem; ferramentas de suporte a método; ferramentas de gerenciamento de interface de usuário; ferramentas de dicionário de dados; ferramentas de edição de diagramas; ferramentas de prototipagem; ferramentas de gerenciamento de configuração; ferramentas de preparação de documentos; ferramentas de edição de texto; ferramentas de estimação e planejamento.

3.5 Exemplos (Taxonomia) de Ferramentas CASE no mercado.

São apresentados abaixo exemplos de ferramentas CASE utilizadas nas diversas fases ou processos no desenvolvimento de um software. Algumas ferramentas talvez já tenham suas versões mais atualizadas, porém, o objetivo é apresentar a diversidade e pluralidade destas ferramentas. Do ponto de vista das ferramentas comerciais, as duas principais suítes do mercado no atendimento ao suporte da

[26] FARIAS, fl. 10.
[27] SOMMERVILLE, 2007, p. 57.
[28] FARIAS, fl. 11.

engenharia de software são: suíte da IBM-Rational e suíte da Borland. Ambas possuem ferramentas integradas para apoio ao desenvolvimento de software.

Alguns aspectos devem ser levados em conta na decisão de escolha de uma ferramenta CASE, como: capacitação; suporte; integração entre ferramentas. Ferramentas Upper-Case e Ferramentas Lower-Case.

- Processo de Desenvolvimento de Software: Unified Process (UP, RUP, EUP, etc.); Open - object-oriented process environment & Notation; Agile (Extreme programming; Feature driven development; Scrum; Crystal clear methodology; Dynamic systems development method; Catalysis).

- Modelação de processos de negócio: Aris Toolset (*www.idsscheer.com*), Mega Suite (*www.mega.com*), Provision (*www.proformacorp.com*).

- Gerência de Projetos: solução proprietária mais famosa é Microsoft Project. Já do ponto de vista de ferramentas livres temos: Open Workbench; Mrproject.

- Gerência de Requisitos de Software: As ferramentas proprietárias mais famosas são: Requisitepro – IBM/Rational; Caliberrm – Borland; Doors – Telelogic. Já com relação a ferramentas livres é restrito o conhecimento de ferramentas nesta área, apenas iniciativas preliminares de desenvolvimento.

  - Rational Requiste Pro: Organização, visualização e acesso a documentos em editor de texto (Ms Word); Modificação de requisitos (nos documentos ou no banco de dados); Criação e alteração de matrizes de atributos dos requisitos; Criação e alteração de árvores de rastreabilidade; Criação, alteração e visualização de relações hierárquicas entre requisitos; Criação de relações de rastreabilidade entre projetos; Interface Web; Suporta Oracle ou Sqlserver; Indicação de mudanças em requisitos relacionados (rastreabilidade); Integração com: suporte a modelagem visual (Xde, Rose); Suporte a gerência de testes (Test Manager); Suporte a gerência de mudanças (Clearquest); Acesso ao texto do requisito - dentro do documento - a partir da ferramenta de modelagem visual.

- Análise e Projeto de Software: Nas ferramentas comerciais destacamos Xde da Rational e Together da Borland. Ambas as ferramentas possuem recursos avançadíssimos de modelagem visual e competem entre si quanto à funcionalidade superando em larga escala as características encontradas nas ferramentas livres. Temos também o Rose (*www.rational.com*), o Paradigm Plus (*www.cai.com*), o GDPro (*www.advancedsw.com*). Se considerarmos as ferramentas que adicionalmente suportam

abordagens estruturadas temos o System Architect (*www.popkin.com*), o PowerDesigner (*www.sybase.com*) e o Silverrun (*www.silverrun.com*).O *System Architect* é uma ferramenta desenvolvida pela empresa *Popkin Software*. Surgiu inicialmente no mercado como ferramenta de modelagem de software, recorrendo a técnicas estruturadas, quer de modelagem de processos quer de dados. Ao longo do tempo, o *System Architect* foi evoluindo e incorporando outras notações (modelagem de negócio e modelagem orientada a objetos). Suas últimas versões passaram a incluir também a linguagem UMLNesta especialidade duas ferramentas livres se destacam: Argo UML e Umbrello. O Argo UML implementa recursos razoáveis de modelagem visual e possui uma vertente comercial chamada Poseidon. O Umbrello é bem mais limitado pois funciona apenas na interface gráfica Kde. Possui também engenharia reversa limitada e apenas para c++.

- Argo UML: suporta Uml; permite representar 8 dos 9 diagramas; round-trip básico em Java; roda em cima da jvm; exporta para xmi 1.0.
- Xde: suporta UML; suporta j2ee; sincronização automática e manual entre código e modelo; capacidade de realizar modelagem visual UML dentro do ambiente de desenvolvimento (eclipse); possui integração com ferramenta de requisitos; permite a utilização de padrões de projeto; capacidade da criação de patterns personalizados; exporta documentação .html, .pdf ou .rtf; Api que permite acesso aos dados por um programa externo e extensão das capacidades do produto; gera script para a criação do banco de dados; compara o modelo físico com o banco que está em produção acusando diferenças; executa engenharia reversa de dados, a partir do banco de dados ou de código.
- eUML2: é um poderoso conjunto de ferramentas desenvolvidas a partir do zero para o Eclipse. Estas ferramentas são projetadas especialmente para os desenvolvedores para colocar em ação UML ao nível de desenvolvimento: garantir a qualidade do software e reduzir o tempo de desenvolvimento.

- Implementação de Software (apenas ambientes de desenvolvimento em Java): nesta área as ferramentas livres oferecem as opções mais consolidadas do mercado. As ferramentas Eclipse e Netbeans dominam o mercado de ambiente de desenvolvimento para java/j2ee tendo todas as funcionalidades das ferramentas comerciais. Um fato a se notar em relação ao Eclipse e ao Netbeans é que, apesar de possuírem características técnicas muito semelhantes, o Eclipse tem sido escolhido como ferramenta de integração pelas suítes comerciais. Tanto a suíte da Rational quanto a suíte da Borland

promovem a integração das suas ferramentas com o Eclipse. Como ferramenta comercial, o Borland Jbuilder é o maior destaque no mercado.

- Testes e Validação de Software: a área de testes é muito abrangente, pois se destacam várias fases e momentos intensos - funcionalidade, stress, carga, usabilidade, unidade, integração, etc. Não existe uma só ferramenta livre para todas as necessidades, nem tampouco uma ferramenta para planejamento de testes. Exemplos de ferramentas Open: Junit e congêneres (teste unitário); Jmeter (funcional e performance); Testmaker (teste de carga). As suítes comerciais possuem ferramentas de teste que possuem funcionalidades semelhantes entre si. Em geral encontra-se no mercado a utilização conjunta de ferramentas comerciais e livres para testes. A ferramenta comercial de maior destaque é o Rational Test Studio, que cobre a maioria dos tipos de teste, inclusive o planejamento.

  - Test Manager: suporta o planejamento, implementação e avaliação de resultados dos testes; implementação através de roteiros de teste; manual ou scripts da ferramenta de automação; integração com Requisite Pro; indicação de mudanças em requisitos que estão relacionados aos casos de teste; relatórios de cobertura de requisitos (casos de teste planejados, implementados e executados por requisito); relatórios de resultados por caso de teste; casos de teste planejados, implementados e executados por requisito por plano de teste; geração e gerência dos dados utilizados para teste; armazenamento e gerência de logs de execução; interface web para execução de casos de testes manuais; integração com ferramentas de automação de testes de regressão e performance.

- Gerência de Configuração e Mudança: existem ferramentas livres maduras para gestão de configuração e mudança em software livre. Dentre as quais podemos destacar o Bugzilla e o Gnats para gerencia de mudança e o Cvs e Aegis para gerência de configuração. O Bugzilla é preferido por todos os desenvolvedores que utilizam o ambiente Sourceforge enquanto o Gnats é uma iniciativa do GNU. Ambos possuem funcionalidades semelhantes, sendo o Bugzilla mais maduro. O Cvs é a mais antiga e madura ferramenta de controle de configuração em software livre. Ele é bastante utilizado em todo o mercado estando bem maduro para uso. As suítes comerciais possuem as ferramentas Clearcase e Clearquest da Rational para configuração e mudança respectivamente, e a ferramenta Starteam da Borland suportando ambas as atividades. O destaque principal em relação às ferramentas comerciais continua sendo a integração entre ferramentas, fato que inexiste entre essas ferramentas livres.

3.6. Engenharia de ida e volta.

Os modelos, dentro do desenvolvimento de software, facilitam muito no entendimento do sistema que está sendo desenvolvido, por ter o conceito de abstração sempre presente, sendo melhor compreendido as complexidades existentes. Com os modelos, a simplificação da realidade do desenvolvimento do software deixa mais simples o processo, permitindo visualizar melhor o sistema.

Uma das etapas de maior importância no desenvolvimento de software e que dependem dos modelos é a etapa da manutenção. E compreender o sistema neste momento é fator mais do que primordial, pois entender o código-fonte e todo o funcionamento do sistema é passo para saber onde está o erro e assim corrigi-lo. A integração (comunicação) entre o modelo e o código-fonte também é fator primordial. Assim, a engenharia de ida e volta (RTE – Round-Trip Engineering) tem papel de grande importância neste processo de comunicação entre o modelo e o código-fonte. Compreender o sistema, ter toda a comunicação entre modelo e código-fonte terá uma manutenção bem mais eficaz e visível aos olhos do engenheiro de software neste processo.

> A engenharia de ida e volta é uma técnica que se propõe a contornar esse problema. Ela permite a sincronização entre código e modelo: quaisquer alterações do código fonte são sincronizadas de volta para o modelo e vice-versa, de forma que o código fonte permanece consistente com o modelo [Angyal et al., 2006]. Seguindo esse raciocínio, Angyal e outros [Angyal et al., 2006] também afirmam que a engenharia de ida e volta melhora a qualidade do software e a eficácia do desenvolvimento. (MAGALHÃES, 2011, f. 27).

A RTE não pára só por aí, mas tem outras vantagens também. O apoio ao desenvolvimento de software por meio de iterações, diferente das fases seqüenciais, é outra grande vantagem. Na iteração existe assim o envolvimento de todas as fases e "uma parte do projeto é implementada como um arquivo executável em uma iteração" (MAGALHÃES, 2011, f. 27). Com isso, com as iterações a automatização da sincronização destes dois pilares (código-fonte e modelo) fica evidente, necessário e de muita ajuda no processo de desenvolvimento e de manutenção do software.

Outras características (vantagens) da RTE são: capacidade de sincronização dos artefatos existentes; atualização automática dos artefatos em resposta às inconsistências detectadas automaticamente; evolução simultaneamente dos artefatos.

4 APLICAÇÃO: FERRAMENTA ECLIPSE.

A instalação e a execução da ferramenta Eclipse no computador são bem simples e fácil de aplicar. Na primeira execução do Eclipse, este pedirá o caminho para um diretório “Workspace”. É neste diretório que o Eclipse salvará as preferências de utilização e também será o lugar padrão para os projetos. Uma mesma instalação do Eclipse pode utilizar diferentes workspaces, com configurações diferentes, por exemplo, para cada usuário.

A Ferramenta Eclipse possui conceitos importantes para um software que desenvolve outros softwares e aplicações, como: integração; plug-ins; *open source*; *framework*; *Eclipse Public License* (EPL); criação de ambientes integrados de desenvolvimento – *integrated development environments (IDEs);* etc.

> A Plataforma Eclipse é um *framework* para construção de IDEs. Ela simplesmente define a estrutura básica de uma IDE. Ferramentas específicas estendem o *framework* e são introduzidas nele para definir uma IDE em particular. De fato, a arquitetura da plataforma permite que um subconjunto de seus componentes seja usado na construção de qualquer aplicação (TRINDADE, 2009, f. 29).

4.1. O Eclipse: uma Ferramenta CASE.

No capítulo anterior deste trabalho foi discutido o conceito de Ferramenta CASE e pode-se, portanto, a partir desta conceituação, classificar o software Eclipse, no conceito de Ferramenta CASE. Como foi visto, uma Ferramenta CASE pode ser definida como um conjunto de técnicas e ferramentas informatizadas (automação) que auxiliam o engenheiro de software no desenvolvimento de software. O software Eclipse com seus plug-ins de desenvolvimento de softwares, pode ser conceituado da mesma forma. “O Eclipse é um projeto *open source*. Seu propósito é prover uma plataforma integrada de ferramentas. O projeto inclui um projeto principal, que prove um *framework* genérico para integração de ferramentas, e um ambiente de desenvolvimento Java. Outros projetos estendem o projeto principal para suportar tipos específicos de

ferramentas e ambientes de desenvolvimento. Os projetos no Eclipse são desenvolvidos usando Java e executam em vários sistemas operacionais"[29].

4.2. Características do Eclipse e aplicação de alguns plug-ins.

A ferramenta Eclipse utiliza plug-ins para extender suas funcionalidades, que passam a ser componentes específicos da ferramenta Eclipse. A ferramenta Eclipse pode ser configurada com um ou mais plug-ins interligadas entre si, reusados e estendidos entre eles. Um plug-in possui todos elementos necessários para sua execução, como: imagens, textos, código Java, etc.

A ferramenta Eclipse é conhecida pela variedade de plug-ins que constituem sua funcionalidade. Esta ferramenta quando instalada já traz consigo alguns plug-ins instalados. Outros poderão ser posteriormente instalados a partir de outros sites. A instalação dos plug-ins costuma ser, em geral, bastante simples, bastando que o Eclipse seja fechado e o plug-in copiado para o diretório "plug-ins" dentro do diretório onde o Eclipse foi instalado. Alguns plug-ins pedem ainda que alguns diretórios sejam copiados para o diretório "features" também dentro do diretório onde o Eclipse foi instalado.

Alguns plug-ins úteis no uso da ferramenta do Eclipse: plug-in de CVS do Eclipse; Visual Source Safe (VSS); Subversion (SVN); StarTeam da Borland; Ant build; JUnit; etc.

Uma integração possível também é entre Eclipse e Poseidon for UML. A arquitetura de Poseidon mostra como a plataforma Eclipse permite que *plug-ins* sejam adicionados a uma ferramenta no intuito de prover mais funcionalidades a ela. Existem alguns pontos fracos no plug-in, como não permitir a definição e execução de transformação entre modelos, mas este fato pode ser contornado se desenvolvemos um *plug-in* para isto e que também esteja na plataforma Eclipse.

> Segundo a Universidade Federal de Pernambuco o Poseidon para UML é uma ferramenta de modelagem a qual é a evolução da ferramenta de código aberto ArgoUML. É projetada para programadores cujas funções são escrever, testar e manter o código atualizado. Alguns dos seus recursos mais importantes são o UMLdoc para HTML e Word 2003, possui integração com o Eclipse IDE e geração de códigos sofisticados em uma variedade de

[29]TRINDADE, Wilson Nevares. *Analisador de Diagramas de Classe UML em Eclipse*. 2009, f. 26.

> linguagens que ajudam a agilizar o processo de desenvolvimento (IMENES, 2006, f. 13).

Alguns recursos do Poseidon para o UML podem ser relacionados, como: Geração de código flexível; Recursos sofisticados para Java; Integração com o Eclipse; Extensões; etc.

4.3. Edição de código no Eclipse e as mudanças no modelo UML.

Aqui iremos utilizar o conceito de Engenharia Reversa (cf. Anexo C) conhecida por possibilitar a recuperação de informações perdidas ao longo do processo de desenvolvimento do software, que partirá do código-fonte para a geração de um modelo, no caso um modelo da UML. Na Engenharia Reversa define-se como atividade de identificação dos componentes do código-fonte desenvolvido e posteriormente criando uma representação em um nível mais alta abstração. Basicamente, duas etapas: extração e abstração de informação.

> Os objetivos da engenharia reversa são múltiplos, por exemplo: i) lidar com a complexidade, gerando visões alternativas; ii) recuperar informações perdidas; iii) detectar efeitos secundários, sintetizando abstrações superiores; e iv) facilitar a reutilização". (MAGALHÃES, 2011, f. 30).

A Engenharia Reversa não faz milagres ou mágicas, pois neste processo de transferência entre o código-fonte para o modelo, especificamente UML, poderá haver perdas de informações (dados). "A engenharia reversa, portanto, requer o suporte de ferramentas em conjunto com a intervenção humana. A combinação dos procedimentos de geração de código e de engenharia reversa permite uma engenharia de ciclo completo, o que significa a capacidade de se trabalhar em modos de visualização gráfica ou textual, enquanto as ferramentas asseguram a consistência desses dois modos (BOOCH *et al*, 2005, p. 16).

De outra perspectiva, Canfora e Di Penta (CANFORAHARMAN & Di Penta, 2007) afirmam que a engenharia reversa pode ter dois grandes objetivos: redocumentação e recuperação de projeto. A redocumentação visa à produção/revisão de visões alternativas de um determinado artefato, com o mesmo nível de abstração, por exemplo, bela escrita de código fonte ou visualização de Gráficos de Fluxo de Controle (CFGs – Control Flow Graphics), enquanto a recuperação de projeto visa à recriação de

abstrações de desenho a partir do código fonte, documentação existente, conhecimento de especialistas e qualquer outra fonte de informação.

4.4. Edição do modelo UML e as mudanças no Eclipse.

Já neste exemplo temos o conceito de Engenharia à Frente (também chamada de Engenharia de Produção), que partirá do modelo (UML) para a geração do código-fonte. A Engenharia à Frente tem como característica o início em uma abstração de alto nível de implementação lógica independente direcionando para a implementação física do desenvolvimento de um software. Aqui o código-fonte é resultado direto do modelo aplicado, no caso um modelo do UML. Basicamente, este tipo de engenharia, especifica o processo tradicional de desenvolvimento de software.

4.5. Instalação de um *Plug-in*.

Neste tópico será apresentado um exemplo (cf. Anexo B) de uso da ferramenta CASE Eclipse, descrevendo um pacote com classes associadas via UML e a geração do código em Java (cf. Anexo D), usando um plug-in UML. Foi utilizada a versão do ambiente Eclipse Galileo 3.5. Como já foi visto anteriormente, o ambiente Eclipse é um framework para construção de IDEs. Ele praticamente define a estrutura básica de uma IDE, no qual ferramentas específicas estendem o framework e são introduzidas nele para definir uma IDE em particular. De fato, a arquitetura do ambiente permite que um subconjunto de seus componentes seja usado na construção de qualquer aplicação. Também será utilizado o plug-in (todo o pacote) do eUML2 Free Edition 3.4.0.20091120 para Galileo/Eclipse 3.5. eUML2 é um poderoso conjunto de ferramentas desenvolvidas a partir do zero para o Eclipse. Estas ferramentas são projetadas especialmente para os desenvolvedores para colocar em ação UML ao nível de desenvolvimento: garantir a qualidade do software e reduzir o tempo de desenvolvimento. Utilizando o conceito de *Eclipse Modeling Framework* (EMF), no qual é um framework poderoso, tem o objetivo de gerar código de aplicações a partir de definições de modelos, com o uso de modelos facilitando a sua construção e análise.

Como já foi visto, no ambiente Eclipse tem uma unidade básica de funcionalidade, ou um componente, que é chamado de plug-in. O ambiente em si e as ferramentas que a estende são ambas compostas de plug-ins. Uma ferramenta simples pode ser composta de um plug-in, mas ferramentas mais complexas são normalmente divididas em vários plug-ins. Cada plug-in contribui com funcionalidades que podem ser invocadas pelo usuário ou reusadas e estendidas por outros plug-ins. O motor de execução da plataforma é responsável por achar e executar os plug-ins. Ele é desenvolvido sobre a Plataforma de Serviços OSGi, que fornece um padrão flexível de componentes no seu framework permitindo que plug-ins sejam instalados e removidos sem a necessidade de reiniciar a plataforma.

O ambiente Eclipse engloba os componentes centrais necessários ao desenvolvimento utilizando a plataforma. Seus componentes são essencialmente fixos e distribuídos como um único pacote. Todo o pacote incluso no ambiente Eclipse é referenciado como Eclipse Software Development Kit (SDK), suportando ambientes integrados de desenvolvimento (IDEs). Será utilizada a linguagem UML e Java, mas o ambiente Eclipse pode ser usado para criar diversas outras ferramentas através de seus diversos plug-ins. Para utilizar as ferramentas acima descritas no exemplo que será apresentado o sistema requer uma versão atual do Java instalada no sistema. Implica assim nas ferramentas de desenvolvimento Java – Java Development Tools (JDT), e o ambiente de desenvolvimento de plug-ins – Plug-in Development Environment (PDE).

A instalação do plug-in pode ser realizado de modo simples, objeto e rápido (cf. Anexo A), porém, com conexão à internet.

4.6. Estudo de Caso.

Para avaliar os recursos do Eclipse com o plugin eUML2 como uma ferramenta CASE, foram implementadas as classes e o código correspondente a uma aplicação utilizada na disciplina Laboratorio de Sistemas na Web, baseado em Eriksson,H.; Penker, B. , Lyons B. e Fado D. (2004, cujo diagrama de classes é apresentado na figura):

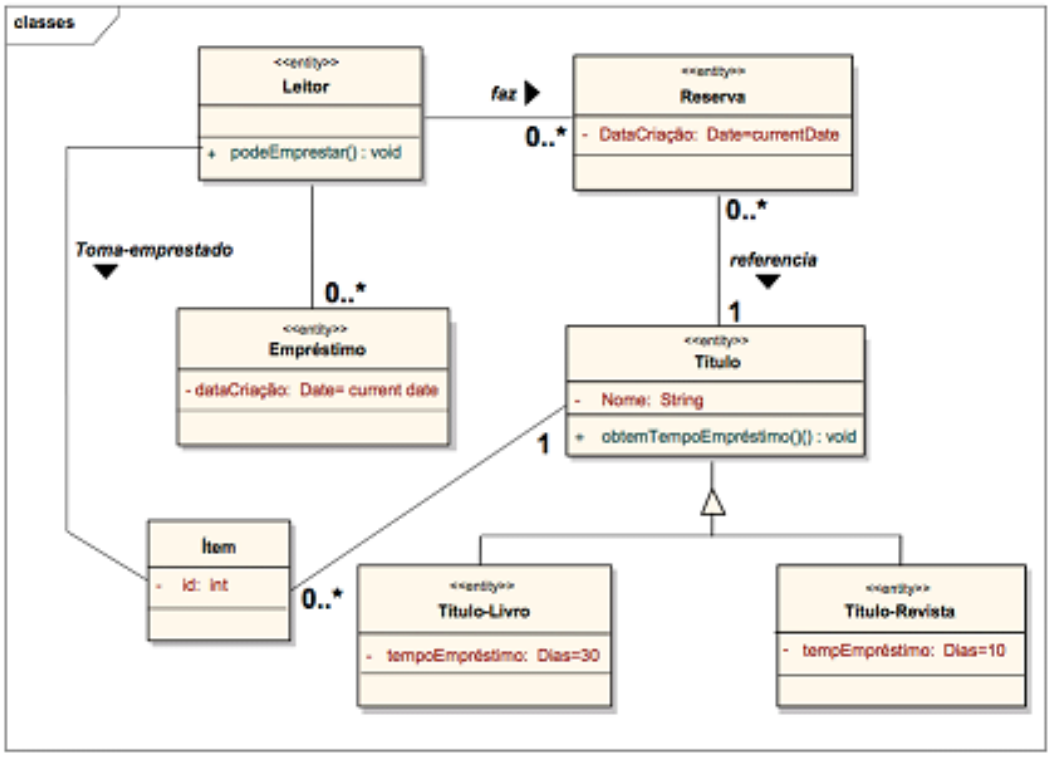


Figura 5 – Diagrama de Classes[30]

[30] FONTE: Guimarães, A. M. Desenvolvimento de Sistemas na WEB. Disciplina do Curso Laboratório de Sistemas na WEB. Notas de Aula. Publicação Interna. Curso de Especialização em Análise de Sistemas. DCC/UFMG. 2009.

5 CONSIDERAÇÕES FINAIS

5.1. Contribuições Práticas e Didáticas do Trabalho

O objetivo da Engenharia de Software é o focar o desenvolvimento de software dentro de custos adequados e de alta qualidade. O desenvolvimento de um software não é um processo tão exato, mas costuma ter complexidades e dificuldades relativas ao entendimento e compreensão do sistema que se deseja produzir. A informalidade no desenvolvimento não ajuda, apenas atrasa os projetos. Planejamento e utilização das ferramentas certas são elementos essenciais no acompanhamento das melhores técnicas para o desenvolvimento do melhor software que atenda a necessidade de cada realidade.

Atualização e integração das ferramentas são elementos complexos, porém necessários (prioritários) na conquista do software de excelência na qualidade. Ambientes, processos e ferramentas devem estar sempre atualizados e reavaliados, para que a qualidade e a produtividade possam caminhar juntas dentro de uma empresa.

Tudo isto deve ser acompanhado de baixo custo no desenvolvimento de um software. Qualidade e produtividade só tem sentido quando existe um baixo custo conforme o gráfico de custo da empresa.

Este trabalho procurou mostrar os elementos centrais para o melhor caminho de desenvolvimento de um software, através dos ambientes, processos e ferramentas em que o mercado está investindo atualmente. Porém, atualização e busca das ferramentas mais eficazes se fazem através de um processo contínuo. Para um engenheiro de software, o presente trabalho ajuda e dá uma visão integrada das principais ferramentas (processos e instrumentos) para o melhor desenvolvimento de um software. Conceitos importantes foram apresentados neste trabalho: Ambientes e processos de Desenvolvimento de Software; Integração; Ferramentas CASE; UML; utilização de um ambiente que integre a UML e a geração de código Java; engenharia ida e volta.

O engenheiro de software trabalhando e integrando bem estes conceitos especificados poderá, em sua empresa, dotar excelentes ferramentas para um crescimento de desenvolvimento de software no mercado, tendo em vista a qualidade e o baixo custo.

5.2. Trabalhos Futuros (uma apreciação pessoal)

Pontos negativos e de dificuldades podem ser elencados na produção deste trabalho, como: a falta de padronização nos ambientes e processos de desenvolvimento de software (uma teorização excessiva); diversidade de ferramentas (versões e atualizações). A diversidade de ferramentas é um dos pontos negativos, pois o próprio trabalho, ao procurar encontrar soluções foi em direção contrária aos objetivos principais no desenvolvimento de um software: alta qualidade e baixo custo. Assim, a diversidade de sistemas, ambientes e ferramentas, hoje presente no mercado, gera um dilema típico em todo engenheiro de software: qual a melhor ferramenta a ser utilizada, ou como integrar estas ferramentas? Diante de tanta informação como produzir o melhor software com o melhor processo sem custo alto?

Essa problemática faz com que o engenheiro de software busque a via melhor para seu trabalho, ou seja, utilize do conceito "integração" para ser o mais efetivo possível no desenvolvimento de um software.

Como ponto positivo, paralelamente como trabalho futuro, é de meta para o engenheiro de software aperfeiçoar e aprofundar o ambiente Eclipse, integrando os vários plug-ins, na utilização das várias necessidades, procurando por melhor material de apoio: tutoriais, manuais, exemplos, resoluções de problemas na instalação e utilização do mesmo.

Os conceitos abordados neste trabalho foram/são de grande importância para um Analista de Sistemas. Pessoalmente, o trabalho me ajudou a rever os vários conceitos numa perspectiva mais elaborada e atualizada, de forma a perceber especificamente o ambiente Eclipse em sua importância (aqui lembro bem, o conceito de integração) com sua dinâmica nos vários plug-ins que existem e a potencialidade de uma ferramenta CASE no desenvolvimento de softwares. A pesquisa e a produção do trabalho foram elementos satisfatórios e necessários na vida de um analista que integra teoria e prática na sua vida acadêmica. Vejo também, que o trabalho abriu portas para vários campos de pesquisa até o momento ainda visto de modo superficial em algumas disciplinas, como: ferramentas CASE, conceito de integração, ambiente Eclipse, Ambientes e processos de desenvolvimento de softwares e outros.

REFERÊNCIAS

ANDRADE, Jeann Marcell SILVA. *Avaliação de Processos de Software em ambientes de desenvolvimento de software orientados à organização.* 2005. 151 fl. Dissertação (Mestrado) – Universidade Federal do Rio de Janeiro. Programa de Pós-Graduação de Engenharia. 2005. Disponível em: ramses.cos.ufrj.br/taba/index.php?option=com. Acesso em: 30 ago. 2011.

Angyal, L.; Lengyel, L. & Charaf, H. 2006. *An Overview of the State-of-The-Art Reverse Engineering Techniques.* 7th International Symposium of Hungarian Researchers on Computational Intelligence.

BERTOLLO, Gleidson. *Definição de Processos em um ambiente de desenvolvimento de software.* 2006. 118 f. Dissertação (Mestrado) – Universidade Federal do Espírito Santo. Mestrado em Informática. 2006. Disponível em: http://www.inf.ufes.br/~falbo/files/DissertacaoBertolloGleidson.pdf. Acesso em: 25 ago. 2011.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. *UML: Guia do usuário.* Rio de Janeiro: Elsevier, 2005. 474 p.

CANFORAHARMAN, G. & DI PENTA, M. 2007. *New frontiers of reverse engineering*. Em 2007 Future of Software Engineering, FOSE '07, pp. 326--341, Washington, DC, USA. IEEE Computer Society.

CHIKOFSKY, Elliot. *Computer-Aided Software Engineering (CASE).* COMPUTER IEEE Computer Society, 1993.

FALBO, Ricardo A.; RUY, Fabiano B.; PEZZIN, Juliana; Rodrigo Dal MORO. *Ontologias e Ambientes de Desenvolvimento de Software Semânticos*. 2004. 16 f. Departamento de Informática, Universidade Federal do Espírito Santo, Vitória - ES - Brasil. Disponível em: http://www.inf.ufes.br/~falbo/download/pub/2004-JIISIC-1.pdf. Acesso em: 29 ago. 2011.

FARIAS, Adalberto Cajueiro. *Ferramentas CASE: Suporte, Adoção e Integração.* Universidade Federal de Pernambuco. Centro de Informática. 33 f. Disponível em: www.di.ufpe.br/~acf/publications/CASETools-report.pdf.gz. Acesso em: 17 set. 2011.

FISCHER, Alan S. *CASE: Utilização de ferramentas para desenvolvimento de software.* Rio de Janeiro: Campus, 1990.

GUEDES, Gilleanes T. A. *UML: uma abordagem prática.* 3. ed. São Paulo: Novatec Editora, 2008. 336 p.

HOFFMANN, Tatiana Miele. *Avaliação da qualidade da ferramenta CASE System Architect baseada na norma ISO/IEC 14102*. 2001. 105 f. Universidade Regional de Blumenau. Bacharelado de Ciências da Computação. Disponível em: http://campeche.inf.furb.br/tccs/2001-I/2001-1tatianamielehoffmannvf.pdf. Acesso em: 10 out. 2011.

HUMPHREY, Watts S. *Managing the Software Process.* California: Addison Wesley Longman, 1990. 494 p.

IMENES, Elison Roberto. *Seleção de Ferramentas CASE.* 2006. 42 f. Curso de Graduação em Ciência da Computação da Faculdade de Jaguariúna. 2006. Disponível em: http://bibdig.poliseducacional.com.br/document/?view=106. Acesso em: 15 out. 2011.

LEFFINGWELL, D. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise (Agile Software Development Series). Addison Wesley. 2011.

MAGALHÃES, Luis Paulo Alves. *Engenharia de Ida e Volta: Revisão Bibliográfica e Avaliação de Ferramentas.* 2011. 144 f. Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais. 2011. Disponível em: http://www.dcc.ufmg.br/pos/cursos/defesas/1363M.PDF. Acesso em: 03 nov. 2011.

MCCLURE, Carma. *CASE is software automation***.** New Jersey : Prentice_hall, 1989.
MIAN, P. G.; NATALI, A. C.; FALBO, R. A. *Ambientes de Desenvolvimento de Software e o Projeto ADS.* 2001. 7 f. http://www.inf.ufes.br/~falbo/download/pub/RevistaCT072001.pdf. Acesso em: 30 ago. 2011.

OLIVEIRA, K.M., ZLOT, F., ROCHA, A.R.C., TRAVASSOS, G.H., GALOTTA, C., MENESES, C.S., *Domain-oriented Software Development Environment*. The Journal of Systems and Software 72, 145-161, 2004.

PAULA FILHO, Wilson de Pádua Paula. *Engenharia de Software:* Fundamentos, Métodos e Padrões. 2. ed. Rio de Janeiro: LTC Editora, 2003. 602 p.

PRESSMAN, Roger S. *Engenharia de Software.* São Paulo: Makron Books, 1995. 1056 p.

QUADROS, Moacir. *Gerência e Projetos de Software.* Técnicas e Ferramentas. Florianópolis: Visual Books, 2002. pp. 502.

ROCHA, Ana Regina Cavalcanti; MALDONADO, José Carlos; WEBER, Kival Chaves (org.). *Qualidade de Software* – Teoria e Prática. São Paulo: Prentice Hall, 2001. 303 p.

ROCHA, A. R. C., AGUIAR, T. C., SOUZA, J. M. TABA: *A Heuristic Workstation for Software development,* In: Proceedings of COMPEURO 90, Tel Aviv, Israel, 1990.

RUY, Fabiano Borges. *Semântica em um ambiente de Desenvolvimento de Software.* 2006. 121 f. Dissertação (Mestrado) – Universidade Federal do Espírito Santo. Mestrado em Informática. 2006. Disponível em: http://www.inf.ufes.br/~falbo/files/DissertacaoRuyFabiano.pdf. Acesso em: 30 ago. 2011.

SILVA, Alberto Manuel Rodrigues; VIDEIRA, Carlos Alberto Escaleira. *UML, Metodologias e Ferramentas CASE*. Edições Centro Atlântico, 2001.

SOMMERVILLE, Ian. *Engenharia de Software.* 8. ed. São Paulo: Pearson Addison-Wesley, 2007. 552 p.

TRINDADE, Wilson Nevares. *Analisador de Diagramas de Classe UML em Eclipse.* 2009. 58 f. Curso de Engenharia da Computação da Escola Politécnica de Pernambuco – Universidade de Pernambuco. 2009. Disponível em: http://dsc.upe.br/~tcc/20091/TCC%20-%20Thiago%20Trindade%20-%202009.1.pdf. Acesso em: 03 nov. 2011.

ANEXOS

ANEXO A – INSTALAÇÃO DO PLUG-IN EUML2.

A seguir temos os passos para a instalação de um Plug-in e demais ferramentas que o acompanham, juntamente com a aplicação do exemplo do caso da ferramenta CASE associando UML.

- Foi instalado o seguinte plug-in: UML2 SDK Tools (eUML2 Free Edition 3.4.0.20091120 para Galileo/Eclipse 3.5). O UML2 Tools pode ser encontrado em "Modeling" no gerenciador de atualizações. Os plug-ins são instalados através do gerenciador de atualização do eclipse: - Para atualizar a instalação existente seleciona-se o menu Ajuda → Check for Updates. O sistema irá procurar e instalar atualizações para os componentes de software disponíveis. Para instalar a nova funcionalidade, selecionar-se Ajuda → Instalar Novo Software. Seleciona-se na lista um site de atualização. - Para adicionar um novo site de atualização selecionar, pressiona-se o botão "Adicionar" e é feita a inserção da URL. Às vezes é preciso desmarcar a opção "Agrupar itens por categoria" - nem todos os plug-ins disponíveis são categorizadas. Se eles não são classificados não serão exibidos.

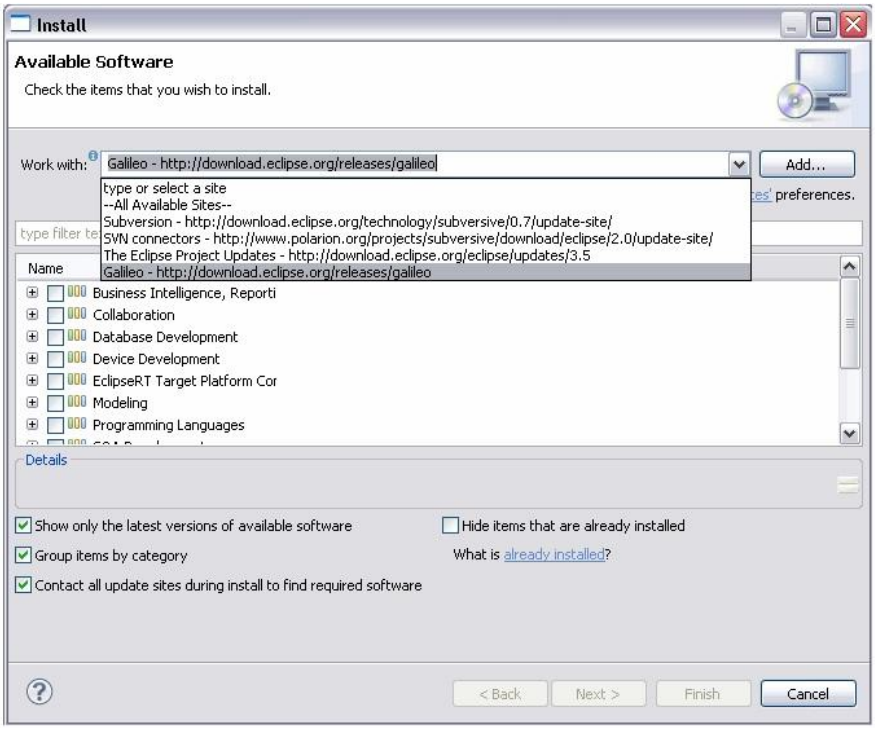


Figura 6 - Gerenciador de Atualização do Eclipse

Fonte: Próprio autor.

ANEXO B – EXEMPLO PRÁTICA DA CRIAÇÃO DE UMA CLASSE NO PLUG-IN EUML2 NO AMBIENTE ECLIPSE. EXEMPLO DE USO DO CASE.

- Criando Diagrama de UML:

O exemplo a seguir irá criar um diagrama de classe com o UML2 Tools. Foi criado um novo projeto java "de.vogella.uml2.first" e uma nova pasta "UML2". Clica-se com o botão direito sobre a pasta "UML2", seleciona-se New → Other. Seleciona-se UML 2.1 Diagramas e então o "Class Diagram" como o tipo.

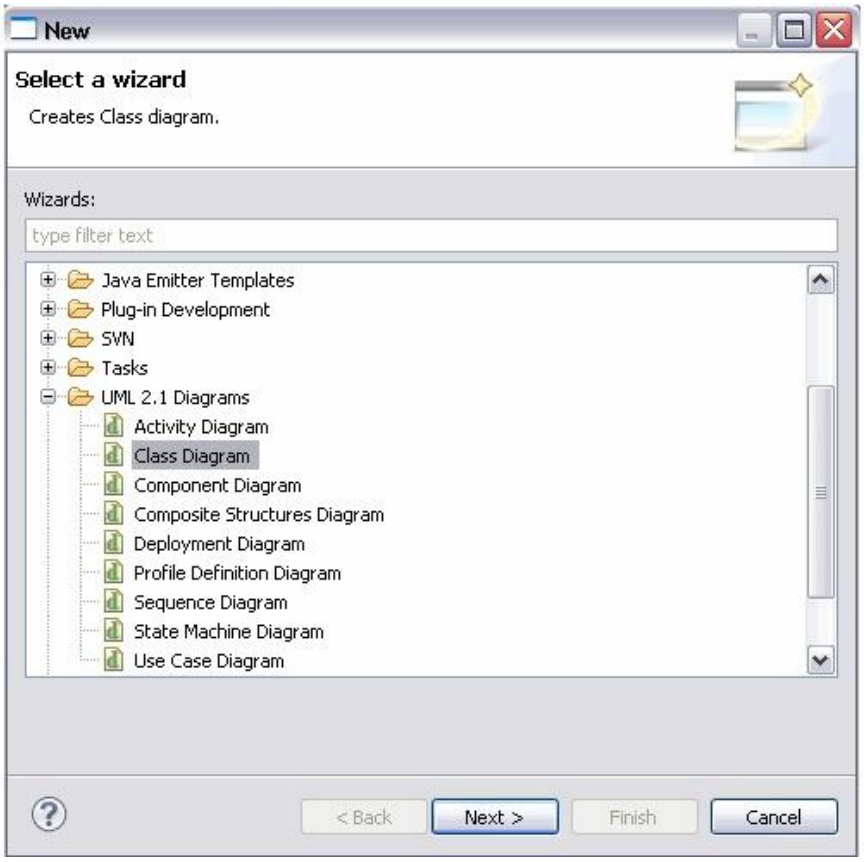


Figura 7 – Criando um Diagrama de Classe no Eclipse.

Fonte: http://www.vogella.de/articles/UML/article.html

- É fornecido o nome do diagrama com um nome específico, no caso será definido: "myclasses.umlclass".

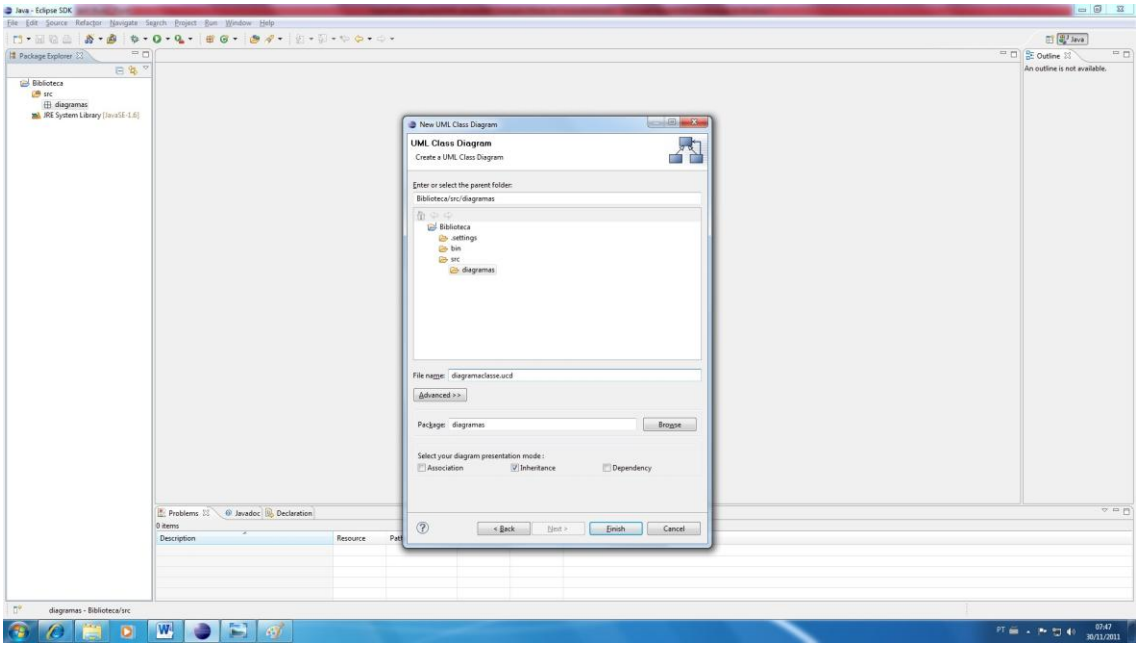


Figura 8 – Definindo o nome da classe.

Fonte: Próprio autor.

- Com a "Palette" (conforme figura 7) pode-se selecionar o tipo de elemento que ajudará na criação do diagrama. Por exemplo, seleciona-se "Class" e clica-se no espaço em branco para criar a representação UML de uma classe. O resultado será uma classe, conforme a figura 8.

Figura 9 - Palette

Fonte: http://www.vogella.de/articles/UML/article.html

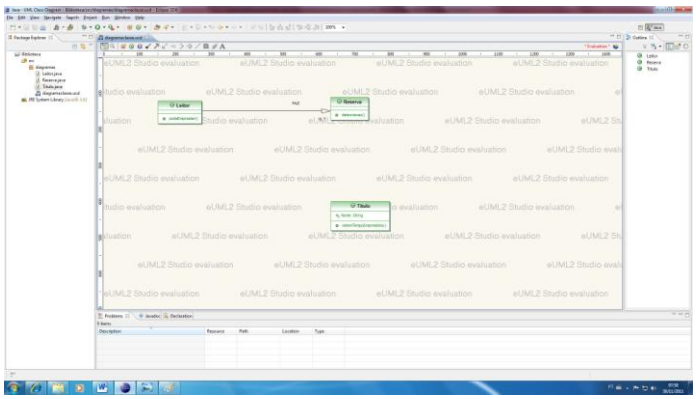


Figura 10 – Class

Fonte: Próprio autor.

- As propriedades de exibição permitem que se altere os atributos dos elementos. Para abrir as propriedades de exibição selecione uma classe com o mouse,

clique direito e selecione “Show Properties View”. Pode-se, então, por exemplo, definir o “Is Abstract” flag como true para criar uma classe abstrata. Se selecionar o elemento de um menu será exibido permitindo que adicione propriedades (campos) e operações (métodos). Conforme figuras 9 e 10.

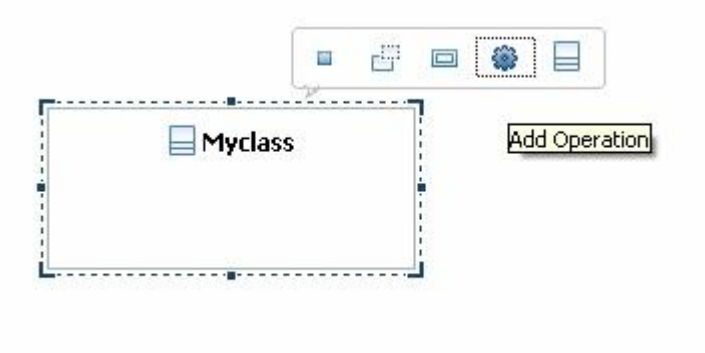


Figura 11 – Myclass

Fonte: http://www.vogella.de/articles/UML/article.html

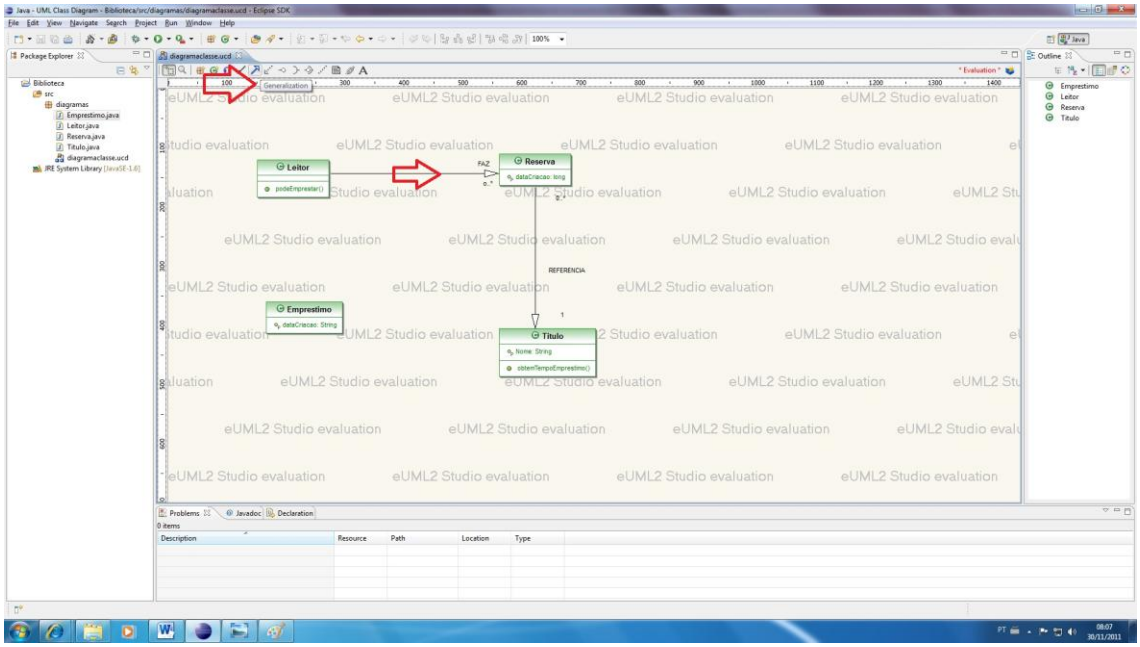


Figura 12 – Criando associações.

Fonte: Próprio autor.

- Note que o diagrama UML atualiza o arquivo *. Uml. No caso de remover o diagrama, pode-se recriar o diagrama a partir do arquivo uml., Através do clique direito do mouse e selecionando a opção “Initialize Class Diagram” no menu (conforme figura 11 logo abaixo).

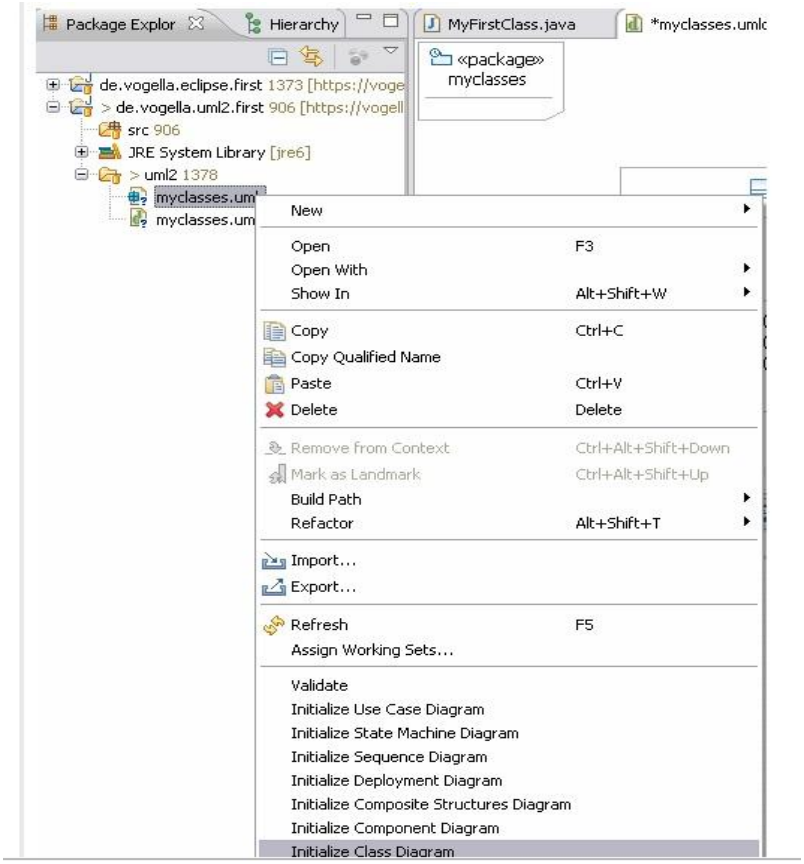


Figura 13 – Initialize Class Diagram.

Fonte: http://www.vogella.de/articles/UML/article.html

- Multiplicidade: pode-se manter a multiplicidade de relação entre duas classes, clicando sobre a associação e através das propriedades “lower” / “upper” na visão de propriedade. (conforme figura 12 e 13).

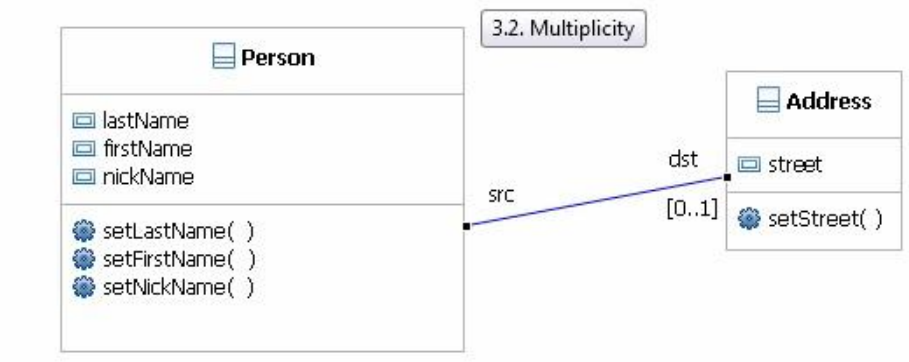


Figura 14 – Multiplicidade.

Fonte: http://www.vogella.de/articles/UML/article.html

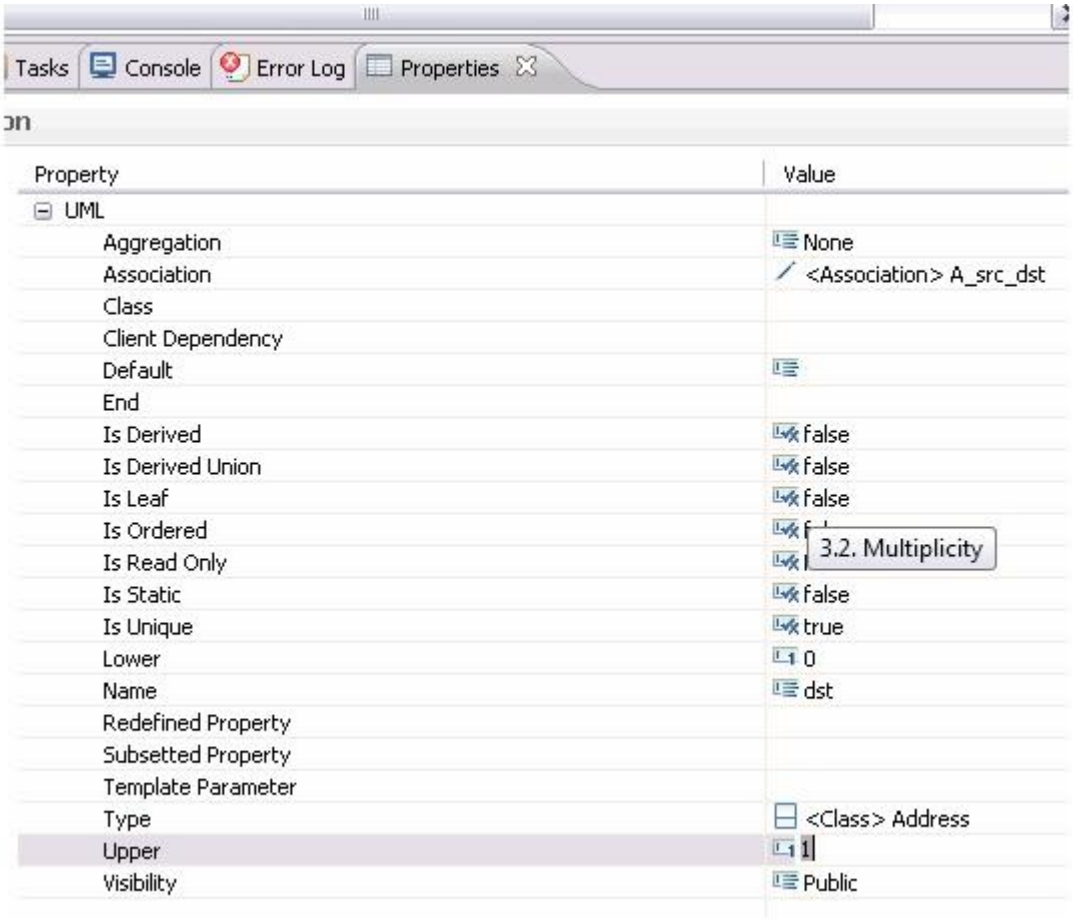


Figura 15 - Propriedades "lower" / "upper"

Fonte: http://www.vogella.de/articles/UML/article.html

- Interfaces: UML2 Tools permite que escolher se prefere usar a notação de " all-and-socket notation" (interface é exibida como um círculo) ou estereótipo UML <<interface>> para a representação de uma interface. Clica-se com o botão direito simples na interface e seleciona-se Show as class" of "Collapse to circle".

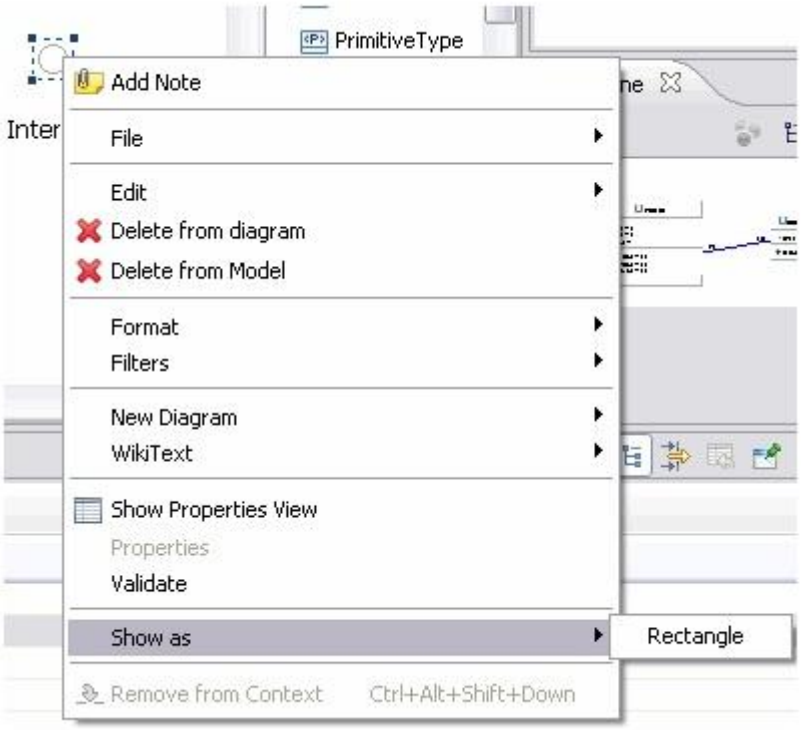


Figura 16 – Interface.

Fonte: http://www.vogella.de/articles/UML/article.html

- Visualização: Investigar o arquivo *. uml. O arquivo *.uml é baseado em Eclipse Modeling Framework (*framework* de modelagem do Eclipse), EMF. Eclipse UML fornece um editor para ele. Alternativa é que pode-se visualizar o arquivo diretamente em um editor de texto.

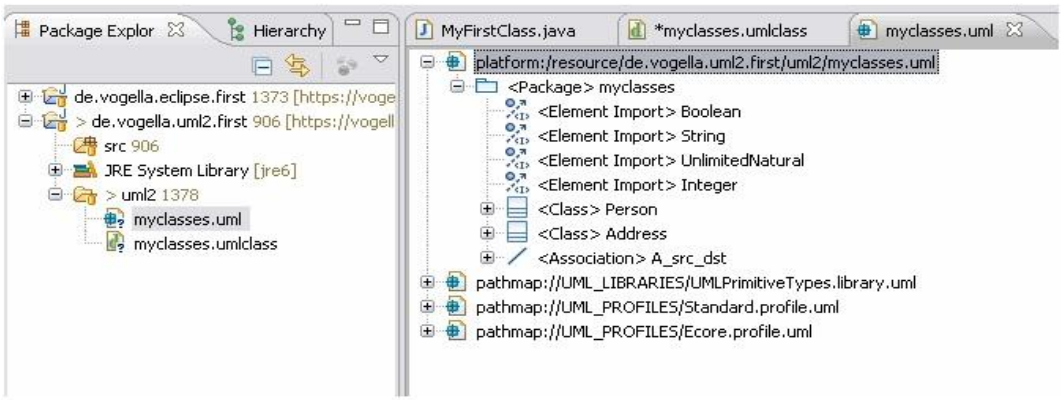


Figura 17 – Visualização.

Fonte: http://www.vogella.de/articles/UML/article.html

- Exportar o diagrama de classes como imagem: Para exportar o diagrama como imagem / gráfico clique a direita no diagrama e selecione File → “Save As Image File...”, como arquivo de imagem.

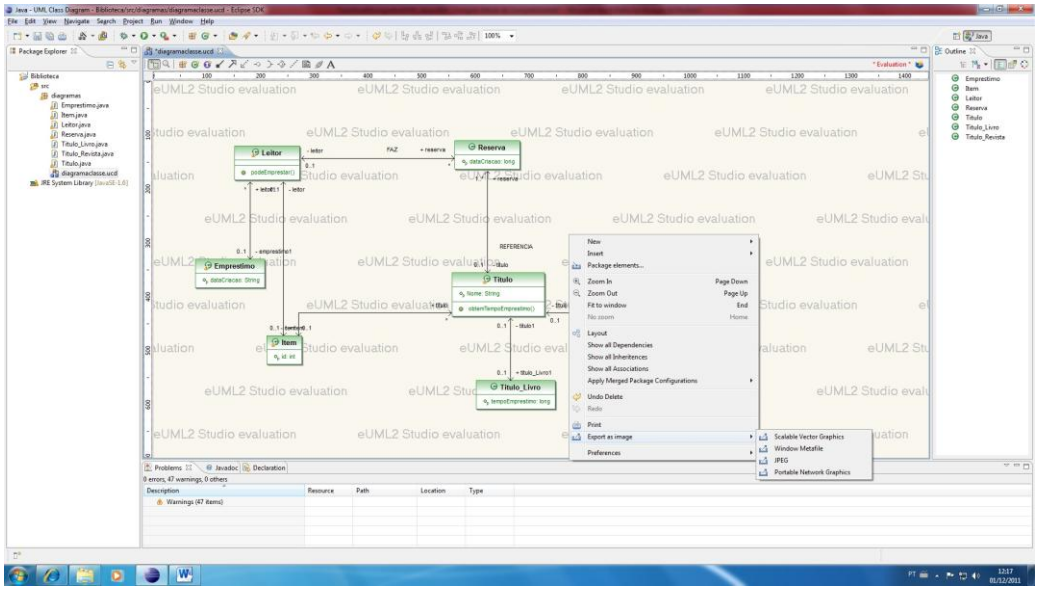


Figura 18 – Exporta o Diagrama de Classe como imagem.

Fonte: Próprio autor.

O plug-in eUML2 é totalmente integrado no JDT. Assim, no JDT, as opções de geração de código Java são usados diretamente por dois componentes em eUML2: Engenharia Reversa e Geração de Código.

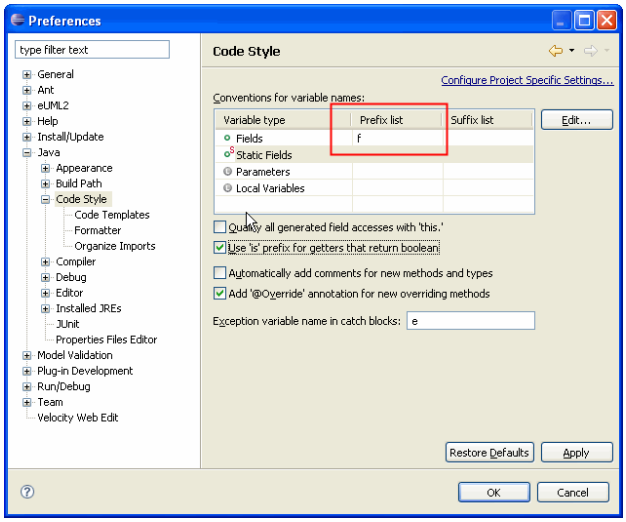


Figura 19 – Preferences do JDT.

Fonte:

http://www.soyatec.com/euml2/documentation/com.soyatec.euml2.doc/preferences/JDT/index.html

ANEXO C – EXEMPLO DE ENGENHARIA REVERSA.

A engenharia reversa UML detecta automaticamente o método getter / setter de um atributo. Leva em conta o prefixo e sufixo de preferências JDT. O nome sem prefixo e sufixo será usado como nome do modelo, chamado como propriedade.

Métodos concretos: se os métodos têm as implementações, a engenharia reversa UML analisar o código para ter certeza de que o método getter retorna este valor de atributo e as mudanças setter neste valor de atributo. Por exemplo, com a definição do prefixo de "f", na classe Java a seguir que contém um nome de atributo = fCompany, o nome da propriedade será "company".

```
public class Employee
{
    private Company fCompany;
    public Company getCompany()
    {
        return fCompany;
    }
    public void setCompany(Company company)
    {
        fCompany = company;
    }
}
public class Company
{
    ...
}
```

Após o processo de engenharia reversa, encontramos UML nome da função = company, em vez de fCompany. O diagrama parecido com este:

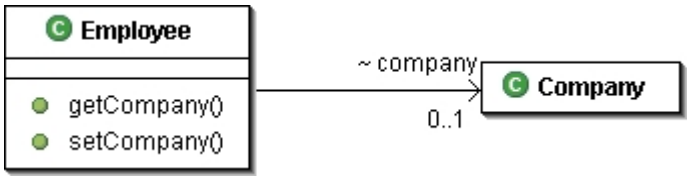


Figura 20 – Exemplo de Engenharia Reversa.

Fonte:

http://www.soyatec.com/euml2/documentation/com.soyatec.euml2.doc/reverse/getter-setter/index.html

E o código ficaria assim:

```
public class Employee
{
    /**
```

```
     * @uml.property name="company"
     * @uml.associationEnd multiplicity="(0 1)"
ordering="ordered"
     *           elementType="company.Company"
     */
    private Company fCompany;
    /**
     * @uml.property name="company"
     */
    public Company getCompany()
    {
        return fCompany;
    }

    /**
     * @uml.property name="company"
     */
    public void setCompany(Company company)
    {
        fCompany = company;
    }
}
```

ANEXO D – GERAÇÃO DE CÓDIGO NO ECLIPSE.

eUML2 usa esta opção para gerar o código Java para atributos. Quando se cria um atributo via diagrama, o gerador de código do eUML2 irá concatenar o primeiro prefixo / sufixo nesta opção para produzir o nome do atributo Java. Usando o exemplo anterior, se adicionarmos um novo atributo String “name”:

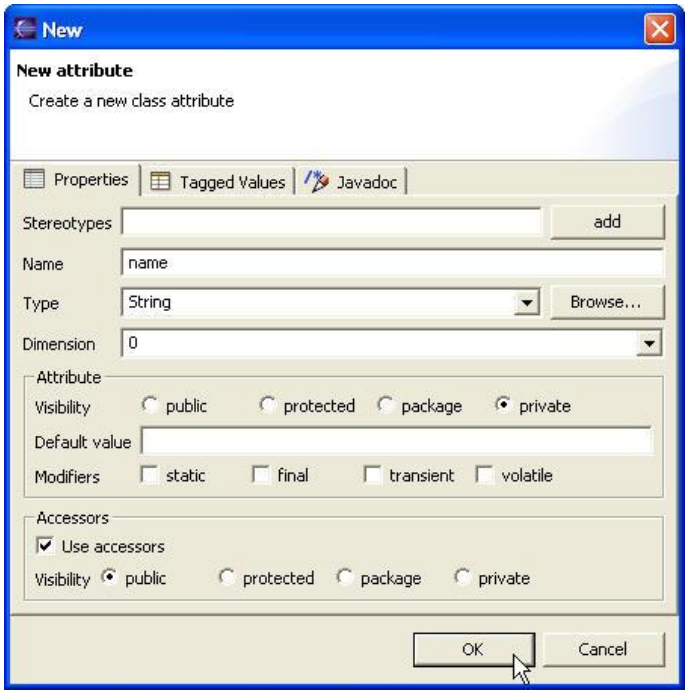


Figura 21 – Classe atributo.

Fonte: Próprio autor.

- Tem-se um fName atributo Java em seu lugar.

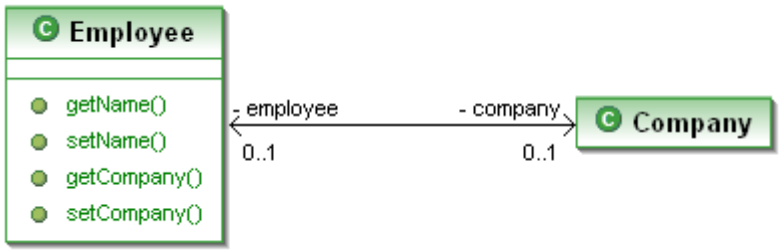


Figura 22 – fName atributo.

Fonte:

http://www.soyatec.com/euml2/documentation/com.soyatec.euml2.doc/reverse/getter-setter/index.html

- Código Gerado.

```
/**
 * @uml.dependency   supplier="demo.Company"
 */
public class Employee {

        /**
         * @uml.property  name="name"
         */
        private String fName = "";

        /**
         * Getter of the property <tt>name</tt>
         * @return  Returns the fName.
         * @uml.property  name="name"
         */
        public String getName() {
                return fName;
        }

        /**
         * Setter of the property <tt>name</tt>
         * @param name  The fName to set.
         * @uml.property  name="name"
         */
        public void setName(String name) {
                fName = name;
        }

        /**
         * @uml.property  name="company"
         * @uml.associationEnd  inverse="employee:demo.Company"
         */
        private Company fCompany;

        /**
         * Getter of the property <tt>company</tt>
         * @return  Returns the fCompany.
         * @uml.property  name="company"
         */
        public Company getCompany() {
                return fCompany;
        }

        /**
         * Setter of the property <tt>company</tt>
         * @param company  The fCompany to set.
         * @uml.property  name="company"
         */
        public void setCompany(Company company) {
                fCompany = company;
        }
}
```

- Geração de Código: As preferências de geração de código são usadas para personalizar o código gerado eUML2. Uma opção está disponível: “Use Velocity Java code template”.

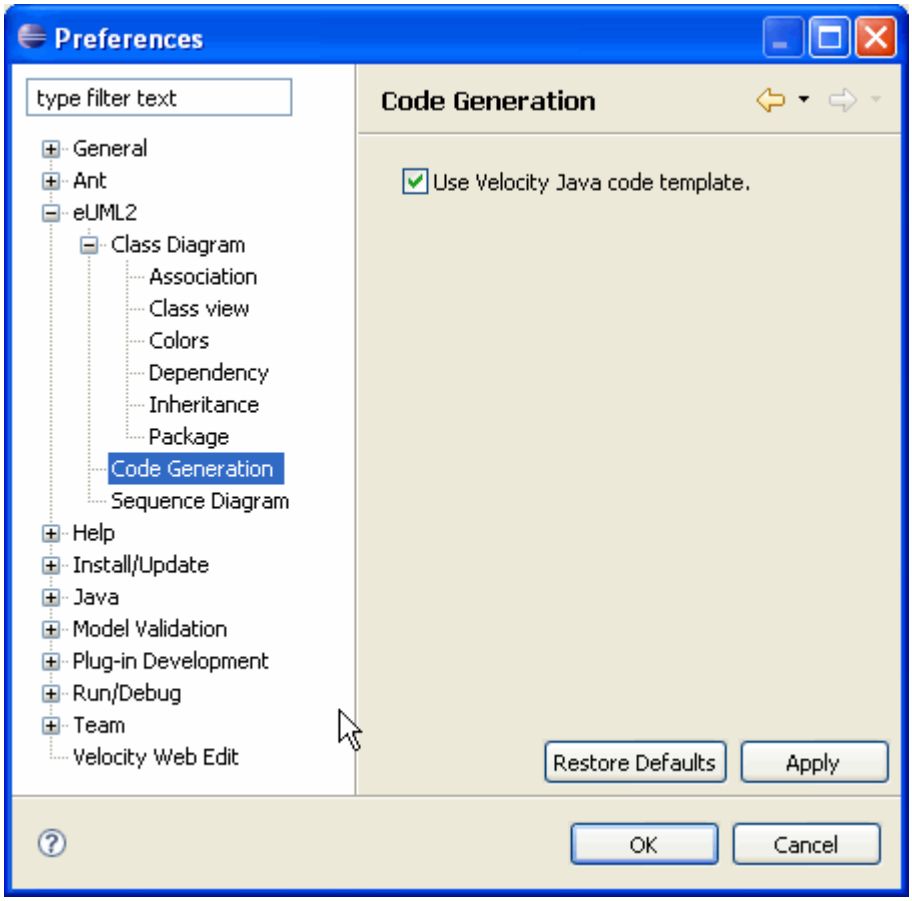


Figura 23 – Code Generation.

Fonte: Próprio autor.