Construção completa de um Caso de Uso com a utilização da UML, programação PHP e padrão MVC

José Gladistone Rocha

Resumo: O objetivo desse estudo é realizar, na prática, e de forma didática, a construção de um Caso de Uso que faz parte de um Sistema de Informação. Sua implementação será em linguagem PHP. Será utilizado o padrão arquitetural MVC. Como ferramenta de Orientação a Objetos será empregada técnica da UML para a documentação do sistema. Com esse trabalho espera-se que sirva aos desenvolvedores de SI como exemplo a se construir outros sistemais maiores. A motivação para escrever esse artigo reside nos desenvolvedores, particularmente aos mais novos, e assim terem como exemplo o desenvolvimento em pequena, mas de forma completa.

Palavras-chave: UML, Estudo Superior.

Abstract: The purpose of this study is to perform, in a practical and didactic way, the construction of a Use Case that is part of an Information System. Your implementation will be in PHP language. The MVC architectural standard will be used. As an Object Oriented tool, the UML technique will be used for system documentation. With this work it is expected that it will serve the IS developers as an example to build other larger systems. The motivation to write this article is aimed at developers, particularly the younger ones, as an example of how to develop on a small scale in a complete wayP á g i n a | **25**.
Keywords: Software Architecture; MVC; Software Engineering, Unified Process.

1 INTRODUÇÃO

Sistemas de software têm sido mais complexos na sua construção particularmente devido à quantidade de tecnologias que hoje estão disponíveis para utilização (CAMBRONERO; VALERO, 2013).

Este trabalho utilizará as boas práticas na construção de um sistema de TI por meio de técnicas consagradas para o desenvolvimento de um sistema de informação. A ideia é a partir de apenas um UC se construir toda a documentação técnica da UML e posteriormente a sua implantação por meio de linguagem PHP.

Parte-se do princípio de que todo trabalho de levantamento de requisitos já havia sido realizado, bem como a análise de negócio.

2 TRABALHOS RELACIONADOS

Engenharia Dirigida a Modelo (EDM) defende o uso de modelos para representar as decisões de design mais relevantes de um projeto de desenvolvimento de software. Um projeto de desenvolvimento de software EDM envolve a criação de muitos modelos. Cada modelo é usado para descrever, visualizar e observar diferentes Pontos de vista de um sistema em diferentes níveis de abstrações (KAHN, PORRES, 2015)

À medida que o mundo real continua mudando, o sistema de software que o representa precisa ser continuamente mantido e evoluído. Desenvolver e manter um sistema de software tão evolutivo é obviamente difícil. Um processo bem disciplinado e uma boa notação de modelagem são essenciais para controlar as atividades na construção e documentação dos diferentes modelos obtidos em diferentes estágios do desenvolvimento de software. O Rational Unified Process (RUP) emergiu como um processo popular de desenvolvimento de software. Como a notação de modelagem, o RUP usa a UML, que é a linguagem de modelagem padrão de fato para o desenvolvimento de software em uma ampla gama de aplicativos (LIU, JIFENG e LIU, 2004).

O uso da linguagem de especificação UML é muito difundido devido a algumas de suas características. No entanto, os sistemas cada vez mais complexos de hoje exigem métodos de modelagem que permitem detectar erros nas fases iniciais de desenvolvimento. O uso de métodos formais torna possível a detecção de erros, mas o custo de aprendizagem é alto Solórzano, Cuesta e Fuente (2005).

Para Lano, K, Androutsopolous, K; Clark D. (2005) UML é uma notação amplamente utilizada para especificação e design orientado a objetos, e também é um padrão internacional. Juntamente com a Linguagem de Restrição de Objetos (OCL), representa uma fusão de linguagens de especificação gráfica e formal que possui um alto potencial para introduzir os benefícios das técnicas de especificação formal no desenvolvimento de software. Os autores retratam ainda que os modelos UML podem ser sistematicamente transformados para refiná-los para formas mais próximas da implementação em plataformas específicas. Por exemplo, a eliminação das classes de associação (que não são expressível em nenhuma linguagem de programação OO), a eliminação de muitas associações (para refinamento para um modelo de dados relacionais), etc.

Ammour e Desfraya (2006) afirmam que na abordagem de Engenharia Orientada a Modelos, o modelo que representa as diferentes facetas do negócio empresarial tende a se tornar grande. É então difícil de entender e manipulá-lo em processo de software colaborativo. A solução óbvia e clássica aplicada por engenheiros e arquitetos de software é fazer a adequada composição de decomposição dos sistemas. Uma multidão de técnicas são propostas com respeito a este princípio de engenharia, e uma das mais conhecidas é a separação das preocupações desenvolvidas pelas abordagens de design baseadas em preocupações. O princípio aplicado para resolver o problema da complexidade é considerar os modelos comerciais empresariais como uma composição de partes menores e mais reverenciáveis, chamadas interesse.

2 ANÁLISE E PROJETO

O caso de uso a ser trabalhado será o de “Gerenciar Nota Fiscal de Venda” de um determinado Sistema de Informação. Dentro da concepção do desenvolvimento de sistemas, várias etapas foram passadas para se chega até nesse momento de definição do UC, como contatos com o cliente, definição de critérios em geral, definição dos requisitos, criação de artefatos e controles do projeto.

Aqui observa-se que esse UC trata-se de um Create, Retrieve, Update e Delete (CRUD) que envolve quatro requisitos funcionais (SOMMERVILLE, 2011):

a) Vendedor inclui Nota Fiscal de venda (NFV);

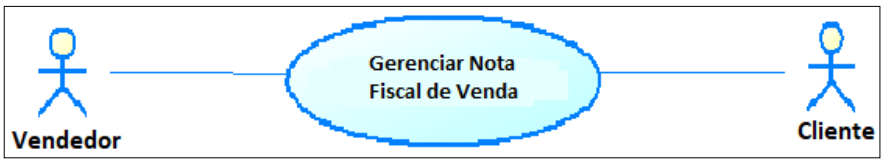
b) Vendedor exclui Nota Fiscal de venda;

c) Vendedor altera dados da Nota Fiscal de venda;

d)Vendedor/Cliente consulta Nota Fiscal de venda

Isso significa afirmar que este UC realiza quatro funções elementares para cumprir suas tarefas dentro de um sistema maior, o qual reside.

Figura 1:Caso de uso Gerenciar Nota Fiscal de Venda



Fonte: o autor

2.1 Descrição do Caso de Uso

Na descrição do caso de uso será usada uma numeração diferente da prevista na ABNT, simplesmente para que as numerações posteriores não se estendam e para trazer maior claridade no entendimento do leitor.

1 Objetivo

Este caso de uso tem por finalidade possibilitar o ator realizar o incluir ou excluir Notas Fiscais de venda e eventualmente fazer alterações em seus dados. Para o cliente como o próprio comprador eles podem necessitar consultar dados da Nota Fiscal.d

2 Ator do caso de uso

Principal: Vendedor

Secundário: Cliente

3 Pré-condição

a) O ator deverá estar autenticado no Sistema.

4 Pós-condição

A Nota Fiscal foi incluída, excluída, alterada ou visualizada no sistema

5 Fluxo Básico

a) o Sistema encontra-se apresentado na página inicial e apresenta as opções de incluir, Excluir [FE1], Alterar [FA2], consulta [FA3] para Nota Fiscal de Venda.

b) O ator seleciona incluir nota fiscal de venda.

c) Sistema apresenta tela com os campos necessários

d) Ator preenche os campos [FE1] e submete ao sistema

e) sistema verifica se todos os dados obrigatórios foram inseridos e caso positivo é gravado no sistema e envia mensagem de sucesso; do contrário uma mensagem é retornada ao cliente indicando tal situação.

6 Fluxos Alternativos Excluir Nota Fiscal de Venda

6.1 Sistema encontra-se na tela para exclusão de NFV

6.2 Ator seleciona uma NFV para ser excluída e realiza sua exclusão.

6.3 sistema emite mensagem de sucesso.

6.4) fluxo volta para a tela inicial do sistema caso o ator deseje sair da tela onde se encontra.

6.1 Sistema encontra-se na tela para alteração de NFV

6.2 Ator altera os dados necessários e salva nos sistemas

6.3 sisteme emite mensagem de sucesso.

6.4 fluxo volta para a tela inicial do sistema caso o ator deseje sair da tela onde se encontra.

7 Fluxos Alternativos Consultar dados de um NFV

7.1 Sistema encontra-se na tela de NFV

7.2 Ator digita dados de um certo campo [FE1] e submete a pesquisar

7.3 o Sistema encontra a NFV e mostra seus dados na tela; do contrário sistema não encontra NFV e emite mensagem de insucesso.

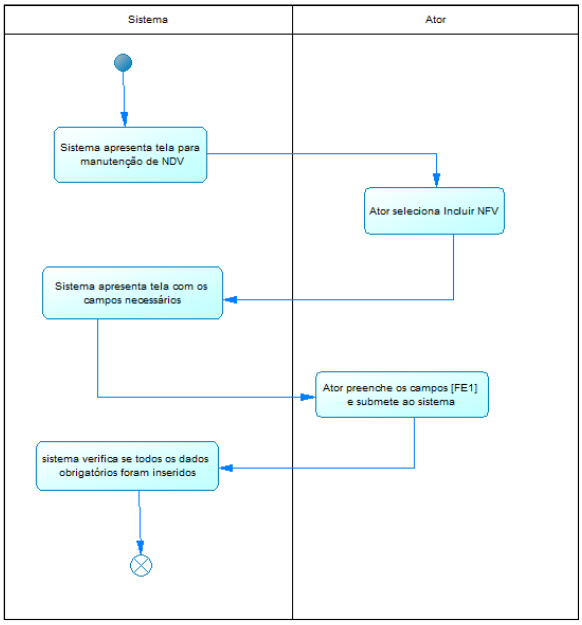
8 Fluxo de Exceção 1

8.1 O sistema encontrou algum dado a digitado faltando ou encontrando algum erro no dado digitado. O sisteme emite mensagem relatando o fato.

2.2 Diagrama de Atividade do UC

O diagrama de atividade é uma excelente ferramenta para se ter uma ideia de como o caso de uso se comporta. Por ela, pode-se observar com é o comportamento entre o ator e o sistema onde há um “diálogo” entre ele. A Figura 2 apresenta o diagrama de atividade do UC trabalhado.

Figura 2: Diagrama de atividade do UC Gerenciar NFV



Fonte: o autor

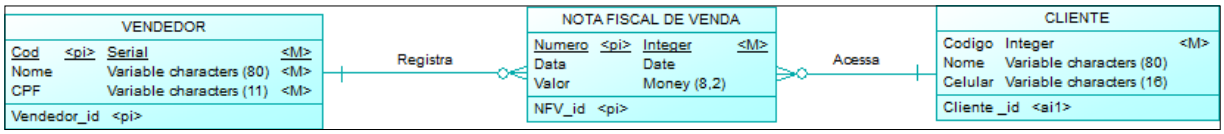
2.3 Projeto de Banco de dados

O projeto de banco de dados se inicia com base na interação com os *stakeholders* do projeto do Sistema de Informação ou utiliza-se do levantamento de requisitos adquiridos pelo Analista de Requisitos.

Após a aquisição dos requisitos passa-se para a construção do Diagrama Entidade-Relacionamento, apresentado a seguir.

Figura 3: Diagrama Entidade-Relacionamento - Modelo Conceitual

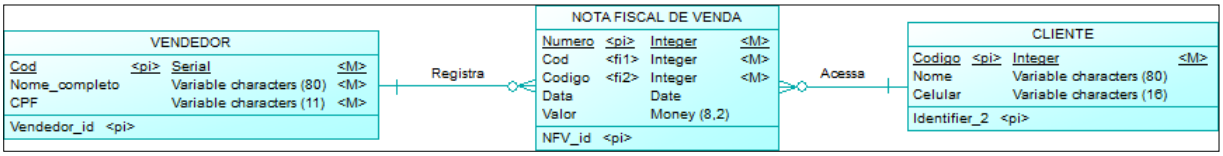
2.3.1 Diagrama Entidade-Relacionamento – Modelo Conceitual



Fonte: o autor

Figura 4: Banco de dados lógico

2.3.2 Diagrama Entidade-Relacionamento – Modelo Lógico



Fonte: autor

Após a construção do modelo conceitual de dados ter sido validado pela equipe de desenvolvimento e *Streakholders*, pode-se transformá-lo para o modelo lógico conforme indicado na figura 4.

2.3.3 Criação física do banco de dados

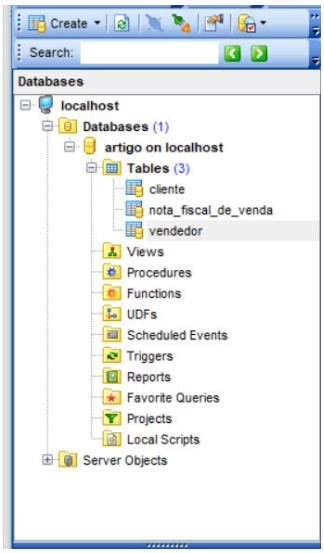
```
/*==============================================================*/
            /* DBMS name:      MySQL 5.0                        */
            /* Created on:     10/06/2017 11:15:04              */
            /
/*==============================================================*/
            create table CLIENTE
            (
```

```
   CODIGO          int not null,
   NOME            varchar(80),
   CELULAR         varchar(16),
   primary key (CODIGO)
);

/*==============================================================
======*/
/* Table: NOTA_FISCAL_DE_VENDA                                  */
/*==============================================================
======*/
create table NOTA_FISCAL_DE_VENDA
(
   NUMERO          int not null,
   COD             int not null,
   CODIGO          int not null,
   DATA            date,
   VALOR           float(8,2),
   primary key (NUMERO)
);

/*==============================================================
======*/
/* Table: VENDEDOR                                              */
/*==============================================================
======*/
create table VENDEDOR
(
   COD             int not null auto_increment,
   NOME_COMPLETO   varchar(80) not null,
   CPF             varchar(11) not null,
   primary key (COD)
);
alter table NOTA_FISCAL_DE_VENDA add constraint FK_ACESSA foreign
key (CODIGO)   references CLIENTE (CODIGO) on delete restrict on update restrict;
alter table NOTA_FISCAL_DE_VENDA add constraint FK_REGISTRA
foreign key (COD) references VENDEDOR (COD) on delete restrict on update restrict;
```

Com base no modelo lógico de dados é possível realizar o modelo físico que se traduz no *script* de geração do banco de dados. O nome a ser atribuído ao banco de dados será “artigo”. A seguir o *script* do banco de dados a ser criado, que será o MYSQL, mas poderia ser qualquer outro.

Após a criação do *script* de criação do banco, deve-se efetivamente criar o banco de dados no MySQL, SGBD conforme havia sido definido anteriormente. A Figura 4 apresenta o banco e as três tabelas do sistema. Poderá ser utilizado qualquer ferramenta para gerenciamento do MySQL;

Figura 5: Banco de dados criado no MySQL

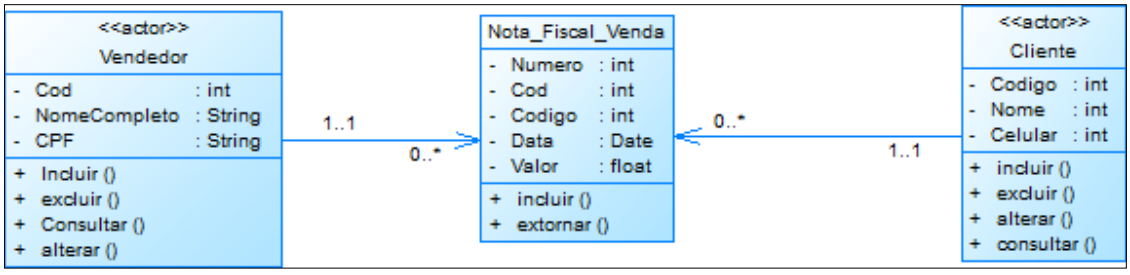


Fonte: o autor

2.4 Diagrama de Classe

Após a análise do sistema, seus requisitos e necessidades, o projetista poderá criar os diversos diagramas de classe, começando pelo diagrama de classe de análise, conforme consta na Figura 6.

Figura 6: Diagrama de Classe de análise



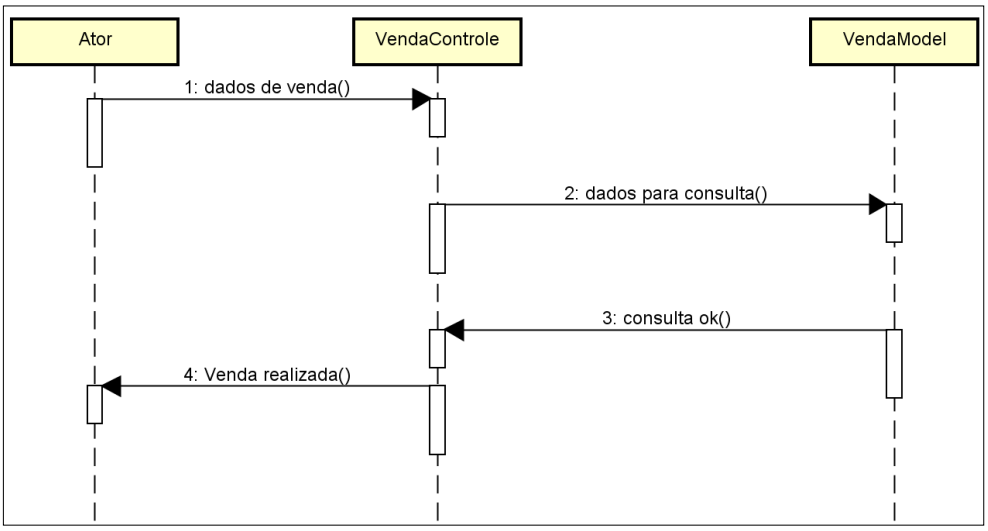
Fonte: o autor

De posse desse diagrama e dos demais artefatos de sistemas o projetista e arquiteto de software poderão definir uma estratégia para programação dos diversos componentes. O mapeamento relacional para este caso será de um para um, ou seja, a tabela Fornecedor liga-se com a classe Fornecedor e assim vale para as duas outras classes, NFV e Cliente.

2.5 Diagrama de sequência

O Diagrama de sequência é útil ao programador para ele conhecer como os objetos interagem entre si em uma determinada tarefa. Muitas vezes esse diagrama não é confeccionado. Ele só é construído para UC mais complexo e fim de facilitar ao arquiteto e programador a lógica aplicadas ao UC

Figura 6: Diagrama de Sequência



Fonte: o autor

2.6 Programação

2.6.1 Classes de objeto em PHP da camada Controller

Classe Cliente

```
<?php
//incluindo classes da camada Model
require_once 'models/VendedorModel.php';
require_once 'models/NFVModel.php';

/**
* Camada - Controladores ou Controllers
* Diretório Pai - controllers
* Arquivo - ClienteController.php
*/
class ClienteController
{
    /**
    * Efetua a manipulação dos modelos necessários
    * para a apresentação da lista de clientes
    */
    public function listarClientesAction()
    {
        if( isset($_REQUEST['in_con']) )
            if( DataValidator::isNumeric($_REQUEST['in_con']) )
            {
                $o_cliente = new ClienteModel();
                $o_cliente = $o_cliente->_list($_GET['in_con']);
                $o_view = new View('views/listarClientes.phtml');
                $o_view->setParams(array('v_clientes' => $v_clientes));
                $o_view->showContents();
            }
    }
}
?>
```

```
<?php
//incluindo classes da camada Model
require_once 'models/VendedorModel.php';

/**
* Responsável por gerenciar o fluxo de dados entre
* a camada de modelo e a de visualização
*
* @package Exemplo simples com MVC
* @author DigitalDev
* @version 0.1.1
*
* Camada - Controladores ou Controllers
* Diretório Pai - controllers
* Arquivo - NFVController.php
Classe NDV

class NFVController
{
    /**
    * Efetua a manipulação dos modelos necessários
    * para a aprensentação da lista de NFV
    */
    public function listarNFVAction()
    {
        $o_NFV = new NFVModel();

        //Listando as NFV cadastrados
        $v_NFV = $o_NFV ->_list();

        //definindo qual o arquivo HTML que será usado para
        //mostrar a lista de nfv
        $o_view = new View('views/listarNFV.phtml');

        //Passando os dados do NFV para a View
```

```
        $o_view->setParams(array('v_NFV' => $v_NFV));

        //Imprimindo código HTML
        $o_view->showContents();
    }
    /**
    * Gerencia a requisições de criação
    * e edição das NFV
    */
    public function manterNFVAction()
    {
        $o_NFV = new NFVModel();

        //verificando se o id do NFV foi passado
        if( isset($_REQUEST['in_con']) )
            //verificando se o id passado é valido
            if( DataValidator::isNumeric($_REQUEST['in_con']) )
                //buscando dados da NFV
                $o_NFV->loadById($_REQUEST['in_con']);

        if(count($_POST) > 0)
        {
            $o_contato->setNome(DataFilter::cleanString($_POST['data']));
            $o_contato->setEmail(DataFilter::cleanString($_POST['valor']));

            //salvando dados e redirecionando para a lista de NFV
            if($o_NFV->save() > 0)
                Application::redirect('?controle=NFV&acao=listarNFV');
        }

        $o_view = new View('views/manterNFV.phtml');
        $o_view->setParams(array('o_NFV' => $o_NFV));
        $o_view->showContents();
    }
}
```

```
?>
```

2.6.2 Classes de objeto em PHP da camada Model

```
Classe Vendedor

<?php
//incluindo classes da camada Model
require_once 'models/NFVModel.php';
require_once 'models/ClienteModel.php';

/**
* Camada - Controladores ou Controllers
* Diretório Pai - controllers
* Arquivo - VendedorController.php
*/
class VendedorController
{
    /**
    * Efetua a manipulação dos modelos necessários
    * para a aprensentação da lista de telefones do contato
    */
    public function listarVendedorAction()
    {
        if( isset($_REQUEST['in_con']) )
            if( DataValidator::isNumeric($_REQUEST['in_con']) )
            {
                $o_Vendedor = new VendedorModel();
                $v_Vendedor = $o_vededor->_list($_GET['in_con']);
                $o_view = new View('views/listarVendedor.phtml');
                $o_view->setParams(array('v_vendedor' => $v_vendedor));
                $o_view->showContents();
            }
    }
```

```
    /**
    * Gerencia a requisições de exclusão de telefones do contato
    */
    public function ecluirVendedorAction()
    {
        if( isset($_GET['in_tel']) )
            if( DataValidator::isInteger($_GET['in_tel']))
            {
                $o_vendedor = new VendedorModel();
                $o_vendedor->loadById($_GET['in_tel']);
                $o_vendedor->delete();

Application::redirect('?controle=Vendedor&acao=listarVendedores&in_con='.$_GET['in_co
n']);
            }
    }
}
?>
```

2.6.3 Arquivo em PHP da camada View

```
<?php
/* Realiza a inclusão dos arquivos com os códigos Model, View, Controller*/
include '/Controller/Controller.php';
include '/View/View.php';
require_once '/Model/Model.php';

/* Pega a ação passada pela URL*/
$acao = $_GET['acao'];

/* Valida a ação passada, verifica se ela existe e se ela e o login
 * Se a ação for exisir e for login inicia a função login do Controller
 * Se não inicia a função login da View*/
```

```
if(isset($acao) && $acao == 'login'){
        $controller = new Controller();
        $controller->login();
}else{
        $view = new View();
        $view->login();
}
?>
```

Após a criação das classes das camadas controller e Model e dos arquivos da camada View o caso de uso está concluído, necessitando que seja testado. Dentro de um projeto de software novos caso de uso serão criados e todo o processo ocorrido com o UC trabalhado será executado, até que se atinja todo o escopo do projeto.

É desejável na fase de testes que seja feita um teste de integração entre todas as unidades já programadas e tidas como prontas, para que haja coesão entre os diversos módulos do sistema construídos.

4 Conclusão e trabalhos futuros

O que se observou foi a criação e execução completa de um caso de uso para que se tenha a noção de como o desenvolvimento de software pode ser executado.

O que se apresentou aqui é uma das várias forma que um software poderá ser desenvolvido. Como trabalhos futuros sereia criação de novos casos de uso para dar uma visibilidade melhor de um sistema de informações.

Referências

AMMOUR ,Samir; DESFRAY, Philippe. “A Concern-based Technique for Architecture Modelling Using the UML Package Merge”. Electronic Notes in Theoretical Computer Science 163 (2006) 7–18

CAMBRONERO, M. E., VALERO,V. “Modelling Distributed Service Systems with Resources using UML”. International Conference on Computational Science, ICCS 2013

KHAN Ali Hanzala; PORRES, Ivan. “Consistency of UML class, object and statechart diagrams using ontology reasoners”.Journal of Visual Languages and Computing

LANO K.; ANDROUTSOPOLOUS, K.; CLARK D. “Refinement Patterns for UML”. Electronic Notes in Theoretical Computer Science 137 (2005) 131–149

LIU, Zhiming, JIFENG He e LIU Jing. ”Unifying Views of UML”. Electronic Notes in Theoretical Computer Science 101 (2004) 95–127

SOLÓRZANO, Manuel; CARLOS, Barrio; CUESTA, E; FUENTE, Pablo de la. “UML Automatic Verification Tool with Formal Methods”. Electronic Notes in Theoretical Computer Science 127 (2005) 3–16

SOMMERVILLE, Ian. Software Engineering. São Paulo: Pearson. 9ª Edição. 2011.