UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM PROCESSO INTEGRADO PARA QUALIDADE EM MODEL-DRIVEN ENGINEERING

DISSERTAÇÃO DE MESTRADO

Marco Antonio Copetti

Santa Maria, RS, Brasil
2012

UM PROCESSO INTEGRADO PARA QUALIDADE EM MODEL-DRIVEN ENGINEERING

por

Marco Antonio Copetti

Dissertação apresentada ao curso de Mestrado em Computação do Programa de Pós-Graduação em Informática, Área de Concentração em Computação Aplicada, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**

Orientadora: Professora Doutora Lisandra Manzoni Fontoura
Co-Orientador: Professor Doutor Osmar Marchi

Santa Maria, RS, Brasil

2012

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

UM PROCESSO INTEGRADO PARA QUALIDADE EM MODEL-DRIVEN ENGINEERING

elaborado por
Marco Antonio Copetti

Como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

Lisandra Manzoni Fontoura, Dra.
(Presidente/Orientadora)

Giliane Bernardi, Dra. (UFSM)

Cristiano Tolfo, Dr. (UNIPAMPA)

Santa Maria, 11 de julho de 2012

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

UM PROCESSO INTEGRADO PARA QUALIDADE EM MODEL-DRIVEN ENGINEERING

AUTOR: Marco Antonio Copetti
ORIENTADORA: Prof. Dra. Lisandra Manzoni Fontoura
Local e data de defesa: Santa Maria, 11 de julho de 2012.

Model-Driven Engineering é o conceito de um novo paradigma de desenvolvimento de software. Esse paradigma prevê que o desenvolvimento de software deixe de ter como artefato central o código e que os modelos e a arquitetura de software tomem esse papel. O conceito não surge sem impactos às estruturas de suporte de desenvolvimento. O modo como o processo de desenvolvimento de software é visto e executado e as organizações relacionadas a ele são obliteradas. A engenharia de software tem se preocupado em expandir os limites da área, criando e adaptando definições, métodos e estruturas para o novo paradigma. A qualidade de software é um dos conceitos de engenharia de software que precisa ser revisto, assim como o processo de desenvolvimento de software. Este trabalho apresenta um estudo sobre os avanços de qualidade de software e do funcionamento do processo de MDE. A partir disso, propõe-se um *framework* de processo de desenvolvimento de software para desenvolvimento em *Model-Driven Engineering*, que integra os conceitos de qualidade investigados e dá visão holística ao desenvolvimento em MDE. O processo foi submetido a uma avaliação conceitual e uma ilustração de uso. Na avaliação o *framework* mostrou abranger os construtos importantes de processo de software. O *framework* objetiva ultimamente integrar e criar sinergia entre as partes participantes do desenvolvimento orientado a modelos.

Palavras-chave: *framework* de processo; processo de software; qualidade de software; *model-driven engineering*; qualidade em MDE

ABSTRACT

Master's Dissertation
Graduate Program in Computer Sciences
Federal University of Santa Maria

AN INTEGRATED PROCESS FOR QUALITY IN MODEL DRIVEN ENGINEERING

AUTHOR: Marco Antonio Copetti
ADVISOR: Professor PhD. Lisandra Manzoni Fontoura
Place and date: Santa Maria, July 11th, 2012.

Model-Driven Engineering a new software development paradigm concept. The paradigm predicts that software development core artifact ceases to be the code and becomes the models and software architecture. The concept does not come without impacts to the supporting structures of development. The way the software development process is seen, executed and structured is obliterated. Software engineering has been expanding the its boundaries, creating and tailoring settings, methods and structures to the new paradigm. Software quality is one of the boundaries of software engineering that is expanded, as well as the software development process. This paper presents a study on the evolution of software quality and on the MDE process. Based on this, we propose a process framework for software development in Model-Driven Engineering, integrating quality concepts that were investigated and giving a holistic view to MDE process. The framework was subject to a conceptual evaluation and had its use illustrated. The framework evaluation showed that the proposed framework is adequate, covering all constructs suggested for a good software development process. The framework ultimately aims to integrate and create synergy between the parties participating in model-driven development.

Keywords: process framework; software process; software quality; model-driven engineering; model-driven quality

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ATL	*ATLAS Transformation Language*
CASE	*Computer-Aided Software Engineering*
CBSE	*Component Based Software Engineering*
CIM	*Computation Independent Model*
DIT	*Depth of Inheritance Tree*
EPF	*Eclipse Process Framework*
GQM	*Goal/Question/Metric*
IBM	*International Business Machines*
M2M	*Model to Model*
M2T	*Model to Text*
MBE	*Model-Based Engineering*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*
MDE	*Model-Driven Engineering*
MOF	*Meta Object Facility*
MTF	*Model Transformation Framework*
NCL	*Number of Crossing Lines in a diagram*
NCU	*Number of Classes per Use case*
NSA	*National Security Agency*
NUC	*Numbeer of Use cases per Class*
OMG	*Object Management Group*
OO	Orientação-Objeto
OpenUP	*Open Unified Process*
PIM	*Plataform Independent Model*
PSM	*Plataform Specific Model*
RUP	*Rational Unified Process*
UML	*Unified Modeling Language*
UTR	*Unified Transformation Representation*
VIATRA2	*VIsual Automated model Transformation*
xUML	*Executable Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

O processo de desenvolvimento de software está em constante mudança. Existem sempre novas ideias e melhores práticas a serem desenvolvidas e testadas, novas linguagens de programação e modificações naquelas existentes; diferentes *frameworks, workflows* e abordagens de como gerenciar projetos, desenvolvimento, avaliação e testes são apresentadas. *Model-Driven Engineering* (Engenharia Orientada a Modelos) surge como uma das mudanças naturais desse processo. Ainda que o conceito não seja novo, somente na última década que se atingiu a estrutura necessária para o conceito evoluir (SCHMIDT, 2006).

Model-Driven (em tradução livre, orientado a modelos) é um termo que descreve um processo onde o modelo é o artefato principal do desenvolvimento ao invés de códigos. O código dentro desse conceito é um produto de trabalho gerado a partir de uma ferramenta que transforma o modelo em um sistema. Essa abordagem tem como objetivo aumentar o reuso em alto-nível, diminuir erros humanos em código durante o desenvolvimento e diminuir custos financeiros e em tempo. Essa mudança não surge sem impactos.

A mudança central no ciclo de desenvolvimento acaba criando um efeito dominó pelo processo de desenvolvimento, gerando necessidade de adaptar o processo a novos conceitos. Sendo assim, existe um impacto na forma em que o processo de desenvolvimento é percebido e executado, assim como as estruturas que dão suporte a esse processo. *Model-Driven Engineering* (MDE) é o conceito *Model-Driven* que abrange todas as áreas da engenharia de software.

Uma das áreas abrangidas é a de qualidade de software. Ao mudar as estruturas básicas de desenvolvimento, as formas com que os desenvolvedores abordam a garantia de qualidade, assim como a forma de avaliar a qualidade e os modelos de implementá-la, modificam-se. Existem autores que pesquisaram e desenvolveram modelos e abordagens para vários aspectos da qualidade de software em *Model-Driven Engineering*, porém não existem esforços no sentido de integrar as ideias em um só processo. Esse trabalho tem como objetivo

propor um fluxo que organize e dê uma visão holística ao processo de desenvolvimento em MDE com qualidade.

1.1 Definição do problema

O crescimento e possibilidade das abordagens orientadas a modelo criaram necessidade de vários estudos na área. Nesse sentido, Schmidt (2006) define que é necessário um esforço integrado de toda a engenharia de software para que adapte seus processos e práticas para dar suporte ao novo paradigma. Uma dessas áreas é a qualidade de software, que no modelo tradicional não se foca em modelos ou práticas que são compatíveis com o que propõe o novo paradigma.

Através da revisão e conhecimento da literatura, pode-se afirmar que existem diversos estudos que procuram solucionar esses problemas. Autores buscam desenvolver modelos e métodos que permitam se modelar com mais qualidade, realizar melhores transformações e validar esses dois processos. Porém a integração desses processos não existe. Usualmente os autores definem seu método ou modelo livre de contexto e não dão atenção para a visão holística de qualidade em MDE.

Sendo assim, o problema definido para esta pesquisa é *como podemos integrar os processos e métodos de qualidade para MDE?*

1.2 Escopo e principais contribuições

Esta dissertação apresenta um *framework* de processo de desenvolvimento de software para o paradigma de MDE associando aos estudos de qualidade de software em abordagens orientadas a modelo. A elaboração do *framework* ocorreu com base nas definições dos processos RUP (*Rational Unified Process*) e do OpenUP (*Open Unified Process*). A essas

definições foram adicionados os conceitos encontrados no estado da arte em qualidade de software e estudo de boas práticas para o desenvolvimento em MDE.

A principal contribuição do trabalho é a definição de um processo de desenvolvimento que integre essas práticas. Outras contribuições incluem:

- Levantamento e estudo sobre as práticas de qualidade em MDE;
- Definição de papéis do processo de MDE;
- Estruturação do processo de desenvolvimento MDE; e
- Estabelecimento da relação de artefatos no processo.

1.3 Organização do texto

O texto está organizado como segue. Na Seção 2 é apresentada a revisão da literatura relacionada aos conceitos mais básicos de MDE, passando pelo seu histórico, sua evolução de conceitos e os diferentes aspectos que envolvem o tema.

Na Seção 3 conceitua-se a qualidade de software em MDE. Nele se mostra os modelos de qualidade e avanços encontrados na literatura que serviram de base para a construção do *framework* deste trabalho.

Na Seção 4 apresentam-se os trabalhos correlatos a esse. Busca-se mostrar as relações entre as contribuições desse trabalho a outros encontrados na literatura.

Na Seção 5 são descritos os métodos do trabalho. Nela é caracterizada a pesquisa, delimitado o objeto de estudo e descritos os procedimentos metodológicos aplicados.

Na Seção 6 descreve-se o processo proposto por este trabalho. A descrição inclui os papéis, tarefas, atividades e artefatos de trabalho relacionados ao processo.

Na Seção 7 se conclui esse trabalho. Apresentam-se as contribuições, fazem-se as considerações finais e sugerindo os trabalhos futuros a essa dissertação.

2 MODEL-DRIVEN ENGINEERING

Modelos são usados no desenvolvimento de software tradicional há algum tempo. Normalmente o seu objetivo é capturar a ideia abstrata do sistema que atenda aos requisitos coletados e analisados. Essa abstração é representada através de diagramas e documentação, os quais são usados posteriormente como facilitadores de comunicação entre os desenvolvedores e os *stakeholders* de um projeto. Eles também podem servir como um guia geral da estrutura que deve ter o sistema implementado nas fases de desenvolvimento.

Em uma abordagem *model-driven* (orientada a modelos) os modelos são colocados como o centro do desenvolvimento. Ao invés de existir um ciclo de desenvolvimento tradicional, no qual a equipe implementaria em código aquilo que foi modelado, a nova abordagem traz uma equipe que modela o software e, através de ferramentas, os modelos são interpretados e traduzidos, gerando-se códigos automaticamente para entregar o produto.

Essa abordagem tem como objetivo permitir que os desenvolvedores concentrem seus esforços em níveis de abstração mais altos, não precisando preocupar-se com os problemas do código e evitando suas implicações (por exemplo, erros de implementação, erros na lógica aplicada e dificuldades com a linguagem de programação). Também se coloca que a utilização de MDE permite reuso em alto nível e faz com que se tenha uma economia financeira e de tempo com o projeto (SCHMIDT, 2006).

Esse capítulo conceitua e revisa o conhecimento sobre *Model-Driven Engineering*. Também é objetivo desse capítulo esclarecer os temas de pesquisa sobre o assunto.

2.1 História de MDE

A primeira tentativa em *model-driven* veio na década de 1980 com a *Computer-Aided Software Engineering* (CASE) (SCHMIDT, 2006). As ferramentas CASE e seus estudos

ficaram somente no meio acadêmico, tendo pouco ou nenhum uso prático e comercial. Eventualmente a ideia barrou em dificuldades computacionais daqueles tempos. O autor atribui isso a linguagens de programação da época, que desestimulavam o uso de transformações de modelos, pois não se utilizava reuso ou orientação a objetos. O autor também atribui ao poder de processamento e limitações de hardware que impediam os desenvolvimentos na área devido às necessidades básicas do processo.

Com a evolução da computação nas décadas seguintes, tanto em hardware como em software, muitos dos problemas foram solucionados. Nos paradigmas mais atuais, as linguagens de programação se afastam da linguagem de máquina e se aproximam da linguagem do desenvolvedor (ou linguagem natural). Além disso, faz-se o uso de objetos e de reuso em programação. Também pode ser destacado que o poder de processamento aumentou nos últimos anos (SCHMIDT, 2006). Essas evoluções deram abertura para que o conceito de desenvolvimento através de modelos pudesse ressurgir e funcionar.

Selic (2003) caracteriza o salto de MDE o mais importante desde os compiladores, como já dito. Ele justifica isso dizendo que é o primeiro salto que segue o conceito de compiladores, mudando fundamentalmente o paradigma de desenvolvimento. Ameller e Frach (2009) colocam que enquanto os outros saltos do desenvolvimento de software procuraram aperfeiçoar o dado pelos compiladores, MDE procura fazer o que os compiladores fizeram, isto é, mudando o paradigma de forma fundamental.

2.2 Conceito de *model-driven*

Model-Driven Engineering, conforme já definido, parte do princípio de modelar um sistema em um nível abstrato, no qual existe maior compreensão do que o software faz ou deve fazer e, a partir disso, desenvolver o sistema sem necessidade de codificá-lo, isso é, o sistema só é desenvolvido no alto nível (HOFSTATER, 2006).

Ameller e Franch (2009) citam que o conceito de MDE é uma evolução dos conceitos de abordagens orientadas a modelos. A definição de engenharia orientada a modelos parte do conceito de *Model-Driven Development* (MDD – Desenvolvimento Orientado a Modelos) que surge da definição de *Model-Driven Architecture* (MDA – Arquitetura Orientada a Modelos).

Em 2001, a OMG (*Object Management Group*) publicou pela primeira vez o padrão *Model-Driven Architecture* (OMG, 2001). A especificação da OMG concentra seus esforços em como um modelo pode ser transformado em código automaticamente através de uma máquina. O grupo estabeleceu uma especificação estável que pode ser usada como guia pelos desenvolvedores e pessoas interessadas em aplicar os conceitos de *Model-Driven.*

Em 2003, Mellor, Clark e Futagami (2003) definiram pela primeira vez o termo *Model-Driven Development*. Os autores explicam que o conceito de MDD é a noção que se pode construir um modelo de um sistema o qual pode ser transformado em um sistema real. Ameller e Frach (2009) discutem que a partir dessa definição qualquer programador em linguagens de terceira geração ou posterior seria um desenvolvedor orientado a modelos, já que, ao utilizar um conceito abrangente de modelo, linguagens de terceira geração são modelos que são transformados em linguagem máquina.

A principal contribuição de Mellor, Clark e Futagami (2003) é conceituar *Model-Driven Development* de forma a esclarecer que esse não é ligado a nenhum padrão ou estrutura como OMG ou MDA. Sendo assim, MDD oferece uma flexibilidade maior para que os desenvolvedores trabalhem, podendo definir diferentes processos de desenvolvimento próprios. Fowler (2005) evidencia essas características, colocando que, diferente de MDA, MDD passa a abranger todo o processo de desenvolvimento e não se prende a estruturas.

Em Schmidt (2006), discute-se a evolução das práticas orientadas a modelo. O autor coloca a evolução do conceito desde as ferramentas CASE até as práticas de MDA e MDD. A discussão central do autor, porém, é que as abordagens Orientadas a Modelo são muito mais complexas que simplesmente mudar a forma como um programa é feito. Ao mudar a atividade principal do processo de desenvolvimento é necessário mudar todas as estruturas que apoiam o mesmo. Haja vista essa característica, para o autor é impossível falar de um Desenvolvimento Orientado a Modelos sem que exista uma Engenharia Orientada a Modelos, isto é, não existem formas de mudar o processo de desenvolvimento sem criar uma estrutura que apóie o processo.

Schmidt (2006) então coloca que para que exista uma integração e um esforço que funcione totalmente, é necessário que existam estudos em todos os aspectos de engenharia que possam apoiar a abordagem orientada a modelo. O autor coloca que somente assim a ciência consegue ultrapassar as limitações e criar um processo mais estável e confiável. O autor também destaca a ideia de que qualquer desenvolvedor de terceira geração seria um

desenvolvedor orientado a modelos, porém ele refuta esse conceito colocando que essas linguagens tem um foco diferente das práticas propostas por *Model-Driven Engineering*.

Em MOLA (2012) ainda se caracteriza mais um nível, coloca-se o conceito de *Model-Based Engineering* (MBE) ou Engenharia Baseada em Modelos. O autor caracteriza o MBE como sendo uma versão mais leve de MDE. O conceito é de um processo onde os modelos são parte chave do processo de desenvolvimento, porém ele não é guiado pelos modelos, o que ocorre em MDE.

Utilizando-se dos três primeiros conceitos, Ameller e Franch (2009) classificam-nos da seguinte forma:

- *Model Driven Architecture*: é a especificação OMG (*Object Management Group*) do desenvolvimento orientado a modelos. Foi publicado pela primeira vez em 2000 e a versão atual é de 2003. OMG propõe uma padronização de abordagens orientadas a modelos. Não exige o uso de modelos da UML (*Unified Modelling Language*), porém é necessário uma linguagem de modelos específica conhecida para a criação de metadados;
- *Model-Driven Development*: o termo foi publicado pela primeira vez em 2003. Em sua publicação era descrito como a ideia de que se pode construir um modelo de um sistema que posteriormente será transformado em um sistema real. Segundo essa definição, qualquer linguagem de programação a partir da 3ª geração (FORTRAN, por exemplo) já apresenta tais características. A diferença principal entre MDD e MDA é que aquele não é padrão OMG e apresenta um conceito mais amplo;
- *Model-Driven Engineering*: publicado pela primeira vez em 2006. O termo remete a algo além do desenvolvimento. O pensamento é que a abordagem orientada a modelos não é somente da parte de implementação, mas envolve a engenharia de software e o processo de desenvolvimento como um todo. O conceito é mais amplo e engloba os dois anteriores, fornecendo suporte para análises e decisões no desenvolvimento orientado a modelo.

A relação entre os três conceitos pode ser visto na Figura 2.1. O número refere-se ao ano da primeira publicação do termo.

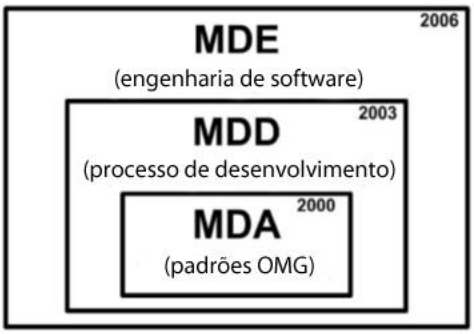

Figura 2.1 - Relação entre MDE, MDD e MDA. (Fonte: AMELLER E FRANCH, 2009 - adaptado)

Os conceitos de MDE podem ser colocados em um paradigma onde os modelos são os objetos primários do desenvolvimento. No MDD, tal paradigma é especificado em fase de desenvolvimento, nele a programação se torna semiautomática, isto é, a partir de ferramentas consegue-se um esqueleto do código dado os modelos, que, como já dito, são os artefatos primários do desenvolvimento. Resta ao desenvolvedor preencher esse esqueleto com o que é necessário para se chegar à solução pretendida pela abstração do modelo (HOFSTATER, 2006; AMELLER E FRANCH, 2009).

Por fim, tendo em vista os conceitos apresentados de MDE, MDD e MDA, pode-se resumir a abordagem como uma mudança de paradigma. Os modelos apresentam uma nova maneira de abordar o desenvolvimento de software e devido a suas características únicas, exigem mudanças nos pensamentos "tradicionais" das várias áreas da computação. Os modelos mostram uma mudança radical no pensamento do desenvolvimento de software, cujos elementos principais são modelos e abstrações visuais.

2.3 Modelos

O conceito de MDE trabalha com uma definição abrangente de modelo. Kühne (2005) e Seidewitz (2003) explicam que a definição de modelo muda conforme o contexto em que se está inserido. Usualmente na engenharia de software o termo modelo refere-se a um artefato de trabalho feito em uma linguagem de modelagem específica.

Em MDE, tal definição é muito específica para ser usada, já que metamodelos e código são considerados modelos também pela abordagem. A definição que se faz de modelo é: descrição de (parte de) um sistema escrito em uma linguagem definida (KLEPPE ET AL., 2003).

Mellor e Balcer (2002) ao propor os modelos executáveis trazem outro detalhe à definição de modelos em MDE. Um modelo para ser executável deve ser descrito com o máximo de detalhes possíveis para permitir uma simulação do sistema, isto é, uma execução viável do modelo. Sendo assim, definições que descrevem modelos como sendo simples ou simplificados devem ser descartados, pois não oferecerão informação suficiente para o processo *model-driven*.

Em OMG (2001) consolidou-se os tipos mais importantes de modelos para as abordagens orientadas a modelo. Nesse caso, as definições foram criadas para servir a *Model-Driven Architecture*, a arquitetura estabelecida pelo grupo. São os três tipos de modelo:

- *Computation Indepent Model (CIM)*: é a visão do sistema de um ponto de vista não computacional. Pode ser chamado de modelo de negócios;
- *Plataform Independent Model (PIM)*: é o modelo que não tem vínculos ou termos que se relacionem a uma plataforma específica. O modelo não remete a nenhum tipo específico de implementação; e
- *Plataform Specific Model (PSM)*: é o modelo que contém detalhes somente dentro de uma plataforma específica. O modelo está intrinsecamente ligado a uma tecnologia e aos métodos de sua implementação.

Exemplificando os modelos, em um sistema de pagamentos: o CIM descreve que o usuário realiza um pagamento; o PIM descreve que o computador ao receber as informações de pagamento, verifica se há saldos, quita a dívida, registra no banco de dados e imprime uma nota; e o PSM descreve que ao receber as informações, o sistema verifica através de um objeto verificaSaldo o saldo do cliente, então utiliza a classe pagamento para pagar, usa o método bancoDeDados para acessar as tabelas no PostgreSQL e modificar os registros, então busca a confirmação no banco e gera uma nota através da classe notaPag e envia para a impressora PrintMais para impressão.

A utilização dessa definição de modelos permite fazer referência as diferentes fases do desenvolvimento de software. Os três modelos passam de um simples reconhecimento da

situação e do problema e detalha até ligar com uma tecnologia de implementação específica. Essa utilização, ressaltando-se o uso de metamodelos, específica para MDA é explicado em Brown (2004).

2.4 Metodologia e processo em MDE

Durante seu período de pesquisa e desenvolvimento, a engenharia orientada a modelos impulsionou a elaboração de uma metodologia e processo de desenvolvimento. Atkinson e Kühne (2003) descrevem a infraestrutura em que MDE trabalha. A infraestrutura se utiliza principalmente de duas especificações referentes a MDE, que são a UML e a MOF (*Meta Object Facility*) (OMG, 2011).

Ameller e Frach (2009) definem essa estrutura como uma arquitetura de metamodelagem em quatro camadas. Atkinson e Kühne (2003) simplesmente classificam como infraestrutura de modelagem. Na Figura 2.2 pode-se observar a infraestrutura.

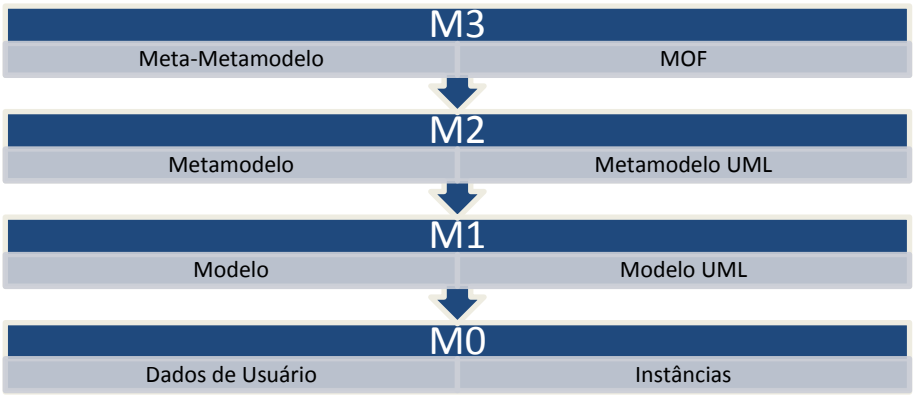

Figura 2.2 - Infraestrutura de modelagem de quatro camadas para MDE. Fonte: Atkinson e Kühne (2003)(adaptado)

As setas indicam que o nível acima descreve o nível abaixo. M3 é auto-descritível, pois é o nível de maior abstração.

Segundo Atkinson e Kühne (2003) o modelo se caracteriza como uma hierarquia de níveis de modelagem, cujo cada nível, exceto o topo, é uma instância do nível acima. O nível

mais baixo, também chamado de M0, é aquele que reúne os dados de usuário. Os níveis são definidos da seguinte forma (da mais alta a mais baixa):

- *M3*: é a camada que contém meta-metamodelos das informações que estão em M2, isso é, ela contém meta-meta-metadados do sistema;
- *M2*: é a camada que contem um metamodelo de M1 que descreve aquilo que os metadados contém;
- *M1*: é a camada que tem o modelo das informações de M0 e consiste dos metadados da aplicação e dos dados do usuário; e
- *M0*: é a camada que contém os dados da aplicação e do usuário. É onde as instâncias e os dados se encontram.

Atkinson e Kühne (2003) colocam que essa estrutura é vantajosa para desenvolvimento, pois consegue acomodar diferentes padrões de modelagem através do MOF e das instâncias que o método realiza.

A infraestrutura de modelagem se relaciona com MDE a medida que a metodologia de desenvolvimento orientada a modelos passa pelos diferentes tipos de modelo. Humm *et al.*(2005) descrevem o fluxo de desenvolvimento orientado a modelos, esse fluxo é revisado e reestruturado por Ameller e Frach (2009). O fluxo pode ser visto na Figura 2.3.

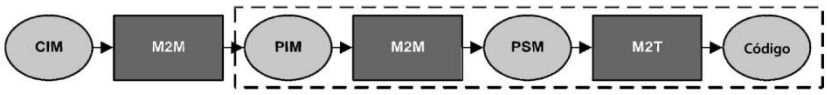

Figura 2.3 - Fluxo de desenvolvimento em MDE. Fonte: Ameller e Frach (2009)(adaptado)

Os quadrados representam transformações. Observa-se que todas as transformações anteriores ao PSM são de modelo para modelo (M2M). Os modelos são transformados e traduzidos para modelos de mais baixo nível e mais específicos até chegar ao PSM. É a partir do PSM que uma transformação de modelos para código pode ocorrer. É importante notar que as linguagens de transformação de modelos para modelos não são as mesmas de modelos para código (OMG, 2001; HUMM et al., 2005).

Ameller e Frach (2009) e Humm et al. (2005) destacam que a utilização das transformações, representadas por M2M (*model to model* – modelo a modelo) e M2T (*model*

to text – modelo a texto) na figura, é conceito chave para MDE. O princípio é que se consiga reutilizar o trabalho feito para um tipo de transformação para todas as outras transformações. Por exemplo, uma transformação criada para inserir critérios de segurança específicos em um software pode ser utilizado para inserir os mesmos critérios em outros sistemas.

Ameller e Frach (2009) descrevem que transformações de CIM para PIM são raras de ocorrer em MDE. Isso pode ser verificado a partir dos trabalhos e estudos práticos na área. Usualmente são utilizados modelos de nível M2 para baixo, a utilização de modelos muito abstratos é rara. Na figura, a parte do processo delimitada pela caixa pontilhada representa o processo que ocorre normalmente.

2.5 Evolução da qualidade de software e MDE

Uma das preocupações do processo *Model-Driven Engineering* é a qualidade. Qualidade de software tem uma conexão profunda com o processo de desenvolvimento de software. Qualidade é avaliada em várias fases, iterações e marcos durante o desenvolvimento. Assim como o software é medido e testado rigorosamente. Linhas de código são analisadas, funcionalidades e requisitos são comparados, e a corretude do software é avaliada.

Pode-se dizer que, conforme Schmidt (2006), MDE é a abordagem que procura criar sinergia entre as partes de engenharia e desenvolvimento de software para produzir melhor com as abordagens orientadas a modelo. A criação de um novo paradigma de desenvolvimento levanta algumas questões sobre o desenvolvimento e como garantir a qualidade nos sistemas desenvolvidos.

Os desenvolvimentos da área, desde que a ideia foi trazida a tona novamente com a publicação de MDA pela OMG (2001), se concentram em alguns tópicos abordados pela abordagem orientada a modelos. A mudança na utilização de termos e preocupações foi evoluindo conforme o conceito evoluiu de MDA para MDE.

Os estudos principais sobre o assunto podem-se classificar como:

- *Modelos*: referindo-se as linguagens utilizadas para modelagem. Criou-se e adaptaram-se metodologias conhecidas para entrar em conformidade com MDE. Também se podem citar (nos parágrafos abaixo) os estudos envolvendo linguagens de transformação;
- *Métodos*: referindo-se a teoria e aos processos. As transformações e o processo de desenvolvimento utilizando MDE são os principais tópicos de estudo; e
- *Corretude*: referindo-se a qualidade dos modelos gerados e as garantias envolvendo os processos de desenvolvimentos que norteiam MDE.

Os estudos de modelos procuraram o que era necessário mudar nos modelos para aplicação em MDE. Mellor e Balcer (2002) é um exemplo desse tipo de estudo. Em seu livro, os autores propõem uma adaptação da UML para MDA, chamada de xUML (*Executable UML*). Em MOLA (2012) também se discute sobre as formas de utilizar UML, modelagem e metamodelagem para utilização em MDE.

Não somente os modelos de software foram estudados como as linguagens de transformação foram estudadas, criadas e aperfeiçoadas para atender as necessidades de MDE. Alguns exemplos são as definições ATL (ATLAS *Transformation Language*) e VIATRA2 (*VIsual Automated model TRAnsformations*) realizadas pela Eclipse. A IBM criou o *framework* MTF (*Model Transformation Framework*) para transformações, assim como a Apache criou Velocity, que suporta transformações de modelo para texto. Informações sobre esses *frameworks* e as linguagens de transformações podem ser encontradas em ATL (2011), VIATRA2 (2011), Demathieu *et al.*(2005) e Velocity (2011), respectivamente.

Os métodos e processos referem-se mais a questões teóricas e de definição de processos em MDE. A evolução de conceitos demonstrada nessa seção surgiu dos estudos nesse sentido. Também surgem nesse sentido as aplicações de MDE e as ferramentas que implementam as linguagens e *frameworks* criados nos estudos de modelos. Pode-se resumir os estudos como as preocupações em evoluir teoricamente o campo e melhorar o cenário de aplicação prática.

Konrad *et al.* (2007) apresentam um exemplo desse tipo de estudo. Os autores propõem um processo (chamado de iMap) para analisar e desenvolver sistemas embarcados através de uma análise em duas camadas utilizando-se MDE. Outra contribuição do modelo dos autores é diminuir os erros durante o processo de desenvolvimento de software.

Albert *et al.* (2010) também demonstram esforços no sentido de métodos em MDE. Os autores propõem uma metodologia para gerar automaticamente esquemas comportamentais para modelos UML e o processo de desenvolvimento. A metodologia procura trabalhar com os aspectos dinâmicos do domínio de um sistema, isto é, trabalhar com os aspectos que mudam durante o desenvolvimento de forma automatizada.

Outros frutos desses estudos são ferramentas para aplicação de MDE. Se nos modelos são definidas as estruturas para o trabalho em MDE, em métodos são as formas de aplicação dos modelos construídos. Exemplos dessas ferramentas são AndroMDA (2012), solução aberta criada para implementações de MDA, publicada pela primeira vez em 2003 e mantida pela comunidade participante do projeto. A IBM também criou sua solução, que representa uma evolução do software *Rational Rose*, chamada *Rational Software Architect* (RSA, 2012). O programa da IBM procura integrar ao desenvolvimento MDE conceitos de suas propriedades intelectuais, como o processo unificado (RUP – *Rational Unified Process*).

Por fim, a corretude vem da necessidade de garantir que a abordagem MDE seja tão confiável e estável quanto o desenvolvimento tradicional. Na medida em que os modelos se tornam os artefatos chaves do desenvolvimento, sua qualidade entra em questão, trazendo estudos sobre a qualidade em modelos. O processo de transformação e os métodos de MDE também precisam ter sua qualidade garantida para que se possam ter transformações precisas e softwares bem construídos.

Assim, pode-se classificar a corretude como qualidade em *Model-Driven Engineering*. Os estudos e desenvolvimentos nesse campo são discutidos no Capítulo 3.

2.6 Considerações

A engenharia de software atualmente coloca grande parte de seus esforços tentando controlar os conceitos principais do desenvolvimento, como, por exemplo, o desenvolvimento de código. O conceito na abordagem orientada a modelos é que toda parte de código será feita pela máquina (considerando algumas diferenças nas diferentes visões do processo). A partir

disso, algumas preocupações aparecem na área. Na medida em que os modelos se tornam o artefato principal do desenvolvimento, a questão é como a engenharia de software tratará essa mudança. A necessidade de um esforço integrado de todas as áreas de engenharia de software para alcançar um processo orientado a modelos que seja confiável motivou a criação do conceito de *Model-Driven Engineering*.

Um estudo na literatura existente em MDE demonstra claramente a evolução do conceito e das preocupações na área. Primeiramente, consolidaram-se as estruturas onde processos orientados a modelo pudessem prosperar. Depois foram feitos estudos dos métodos, do como fazer para que MDE fosse bem sucedido. Por fim, recentemente, é crescente a ideia de como garantir qualidade em MDE.

Apesar de existirem diferentes avanços e estudos na área, ainda não existem processos e métodos que integrem e criem sinergia entre as partes de MDE. Isso é evidenciado pelos estudos de Hutchinson, Rouncefield e Whittle (2011) e Hutchinson *et al.* (2011). Nessas pesquisas, os autores investigaram a utilização prática, principalmente na indústria, de MDE. Foi identificado que as empresas utilizam MDE de forma desconexa. Cada equipe, empresa ou organização que utiliza MDE, cria seu próprio processo e utiliza da sua própria forma, não existindo um método que integre as partes.

Essa necessidade da área também é colocada por Gargantini *et al.* (2009) em uma menor escala. Os autores colocam que é necessária a integração de MDE com outros conceitos e que também é crucial que haja uma integração interna também.

É com base nessa necessidade que esse trabalho se baseia. A proposta de um processo que integre as diferentes partes de MDE. No próximo capítulo, são investigados com maior profundidade os estudos em termos de qualidade em MDE.

3 QUALIDADE EM MDE

Conforme definido no capítulo anterior, a progressão dos estudos e pesquisas em MDE podem ser divididos em três diferentes classificações. Esse trabalho se apoia principalmente nos estudos envolvendo a corretude.

Assim, ressalta-se que a corretude envolve os estudos que procuram criar e ajustar modelos, métodos e processos de forma a garantir que o processo de MDE ocorra de forma estável e confiável. A principal preocupação é possibilitar que a nova abordagem tenha o mesmo grau de confiança e qualidade que existe no desenvolvimento tradicional.

Haja vista esse conceito, pode-se caracterizar a corretude como os estudos de qualidade de software em MDE. Esse conceito, apesar de ser estudado há algum tempo dentro do tema, foi melhor definido por Schmidt (2006) em seu texto. Foi esse autor que propôs que mudassem o termo MDA e MDD para envolver a palavra *engineering*, A partir dessa mudança, o autor passa a incluir toda a engenharia de software na abordagem, procurando assim desenvolver um novo paradigma válido.

Esse capítulo tem como objetivo investigar e demonstrar os estudos de qualidade em MDE que serviram de base para este trabalho. Nas seções a seguir, conceitua-se a qualidade e os aspectos dela que envolvem MDE.

3.1 Qualidade de software

A qualidade de software é uma área dentro da engenharia de software que procura garantir e mensurar a qualidade do software. Para tanto são definidas normas, técnicas, critérios e métodos para o processo de desenvolvimento. Dentro do âmbito da computação, a área se estende por vários pontos do software, sempre procurando como garantir que aspectos possíveis em um software tenham qualidade (KOSCIANSKI E SANTOS, 2007).

Ainda, deve-se levar em conta que a qualidade é uma exigência de negócio dentro do desenvolvimento e cada vez mais empresas se preocupam com a qualidade. É necessário produzir sempre com qualidade ou pelo menos demonstrar que o processo de desenvolvimento tem qualidade (KOSCIANSKI E SANTOS, 2007).

A principal característica da abordagem *Model-Driven* é que os modelos se tornam os artefatos principais do processo de desenvolvimento de software. Os modelos foram criados inicialmente para ajudar os desenvolvedores a construir seus produtos com maior qualidade. O modelo é uma representação gráfica em alto nível dos requisitos, estrutura, comunicações e interações que existem internamente dentro do sistema e junto ao ambiente (LANGE e CHAUDRON, 2005). O desenvolvedor pode mostrar os diagramas aos clientes e explicar como será criada a solução para suas necessidades dentro do software. O modelo é uma planta do que o sistema deve fazer e pode ser usado como um guia nas fases de implementação, sendo úteis para a codificação do programa como visto por quem o modelou (MOHAGHEGHI e AAGEDAL, 2007).

Ainda que esses objetivos estivessem em mente quando foram criados e usados modelos de software como UML, a modelagem de software nem sempre garante a qualidade do produto final. Ainda existe o fato que nem sempre o modelo é feito de maneira correta e com a qualidade necessária. Os arquitetos de software podem ter dificuldades de encontrar a maneira certa de expressar suas ideias em um diagrama ou na documentação (LANGE e CHAUDRON, 2005). Os arquitetos podem criar um grande acoplamento, compromissar a coesão do software, desorientar as interações, criar redundâncias e subestimar o papel de classes nos modelos (GENERO et al., 2004).

Esses problemas em um ciclo de desenvolvimento de software tradicional não são tão problemáticos. Nas fases e iterações de implementação do ciclo, os desenvolvedores podem perceber os erros feitos nos modelos e corrigi-los no código e no modelo. O modelo é mais usado como um auxílio do que uma regra para a programação do software. Um erro feito em um modelo não é final e pode ser avaliado e corrigido facilmente, ainda que existam alguns custos atrelados a isso.

Na abordagem MDE, os erros em um modelo irão criar uma corrente de erros, como a máquina irá interpretar o modelo dado e resultar o produto que foi projetado nele. Um modelo mal feito e de pouca qualidade irá obrigatoriamente resultar em um produto ruim. Rech e Bunse (2008) corroboram com Ameller e Frach (2009) ao colocar que se os modelos se

tornam os artefatos chaves de desenvolvimento, eles passam a ser o alvo de qualidade de software também. Isso se deve ao fato de que em MDE modelos ruins irão gerar software ruim, logo, é importante o cuidado com modelos.

A qualidade em MDE não se limita somente à quão bem foram construídos os modelos do sistema. MDE depende muito das transformações. Se o processo de passar modelos para códigos, às vezes necessitando de transformações de modelo para modelo no caminho, é parte principal do processo de desenvolvimento, torna-se uma preocupação que as transformações tenham sua qualidade garantida (RECH e BUNSEN, 2008).

Somente compreender o processo e procurar construir com qualidade, não irá garantir que as partes tenham qualidade. Portanto, também é necessário validar os modelos e transformações para certificar-se de que implementam realmente a qualidade com que tiveram objetivo.

Sendo assim, pode-se dividir a qualidade em MDE em três diferentes partes. São elas:

- Qualidade em modelos;
- Qualidade em transformações; e
- Validação e controle.

É baseado nessa divisão que essa Seção se apresenta. A seguir serão discutidas cada uma das partes. Esse trabalho também se baseia nessa divisão.

3.2 Qualidade em modelos

Conforme colocado anteriormente, com a mudança do paradigma, modelos passam a ser o artefato principal de desenvolvimento. Autores como Rech e Bunsen (2008) e Nugroho e Chaudron (2008) explicitam que isso fez com que se tivesse uma preocupação maior com a qualidade dos modelos.

Os autores concordam que a modelagem, apesar de guiar os desenvolvedores em um processo de desenvolvimento tradicional, e ter como objetivo a criação de software melhores,

nem sempre ocorre. Punter et al. (2008) colocam que o foco da qualidade de software em um desenvolvimento tradicional é mais voltada ao código e a aspectos do produto final. As boas práticas referem-se a outros aspectos do processo, normalmente que envolvem implicações diretas no sistema resultante.

Assim sendo, como o paradigma coloca os modelos no papel impactante do resultado do sistema final, eles passam a ser preocupação. Ressalta-se que a preocupação com a modelagem correta não é uma ideia que surgiu com MDE. Nugroho e Chaudron (2008) colocam que existiram outros métodos de qualidade em modelos antes que MDE viesse a se concretizar, mas somente na última década que existiram novos estudos na área, os quais maior parte foram impulsionados pela abordagem orientada a modelos.

Na literatura sobre o assunto, existem dois estudos que se destacam. Lange e Chaudron (2005) apresentam um *framework* para modelar com qualidade. Segundo Nugroho e Chaudron (2008), esse foi o estudo que inseriu as preocupações do paradigma dentro da modelagem. Lange e Chaudron (2005) adaptam e utilizam-se de conceitos conhecidos para produzir um modelo compreensível de modelagem com qualidade usando UML.

O segundo estudo que se pode destacar são os objetivos de Mohagheghi, Dehlen e Neple (2009). Os autores em seu trabalho fazem uma revisão da literatura sobre qualidade em modelos, buscando dar um foco ao novo paradigma. A partir de sua revisão os autores propõem um modelo de seis objetivos que devem ser buscados durante o desenvolvimento e modelagem para que se obtenham modelos de qualidade.

Os dois trabalhos são discutidos nas próximas subseções. Ressalta-se que a escolha pelo modelo de Lange e Chaudron (2005) deve-se a sua influência. Quase todos os trabalhos posteriores a esse se referem ao modelo como base ou citam ele sendo utilizado. Mohagheghi *et al*. (2009) conseguem resumir e analisar vários trabalhos sobre qualidade em MDE e a partir disso resumem e adaptam os conceitos de todos trabalhos em um só modelo, que, segundo os autores, é mais simples, prático e evita retrabalho ou redundâncias.

3.2.1 *Framework* de Lange e Chaudron

Lange e Chaudron (2005) propõem um modelo de qualidade de modelos que leva em conta os aspectos da modelagem e do sistema em si.

O modelo leva em conta quatro níveis diferentes (LANGE e CHAUDRON, 2005). No primeiro nível, chamado *uso*, é considerado o uso em alto-nível do modelo. Os usos do modelo podem ser descritos entre três categorias:

- *Operação*: esse uso combina as características de qualidade que são observadas na execução do produto. Isso implica que o sistema já está implementado;
- *Transição*: esse uso combina as características de qualidade que são observadas quando o sistema é colocado em outro ambiente; e
- *Manutenção*: esse uso combina as características de qualidade que são observadas quando o produto é modificado.

Para o modelo de qualidade proposto por Lange e Chaudron (2005) somente o terceiro uso está dentro do escopo do modelo de qualidade proposto pelos autores, já que o primeiro está mais relacionado a como o usuário vê o produto e a transição depende da implementação, não sendo fruto da modelagem do sistema. Os autores por fim introduzem um novo uso, o *Desenvolvimento*, que combinaria as características de qualidade que compreendem os artefatos e partes do produto antes dele ser terminado.

O segundo nível, os autores chamam de *Propósitos da Modelagem*. Isso é o porque um certo artefato é utilizado na modelagem do sistema. Um propósito não se aplica a todos os artefatos, assim como um artefato não tem intenção de se aplicar a todos os propósitos. Os propósitos são listados da seguinte maneira (LANGE E CHAUDRON, 2005):

- *Modificação*: o modelo e o sistema agem como facilitadores de mudanças no sistema. As mudanças podem ser relacionadas a remoção de erros, extensões do sistema ou mudanças ocasionadas pelas modificações nos requisitos do sistema;
- *Teste*: o modelo é utilizado para se criar casos de teste;
- *Compreensão*: o modelo e o sistema são de fácil compreensão. A construção do sistema e suas funcionalidades são colocadas dentro de um modelo de forma a facilitar a compreensão e entendimento correto do sistema em um tempo razoável;

- *Comunicação*: o modelo permite comunicação eficiente sobre os elementos do sistema, seu comportamento e decisões do projeto. Inclui comunicação durante as várias fases de desenvolvimento com diferentes *stakeholders* e documentação para compreensão do sistema em diferentes fases;
- *Análise*: a intenção do modelo é analisar e explorar o domínio do problema incluindo seus conceitos chaves e ajudar na tomada de decisão do projeto;
- *Predição*: o modelo é usado para fazer predições sobre os atributos de qualidade que serão eventualmente implementados no sistema. Esses artefatos ajudariam na melhoria da arquitetura e projeto;
- *Implementação*: o modelo é usado como base para implementação de um código de sistema; e
- *Geração de Código*: o modelo é utilizado para gerar automaticamente um código-fonte para o sistema. A geração de código pode ser completa ou parcial (somente um esqueleto).

O terceiro nível envolve as características de qualidade do artefato. Os autores citam tanto características para os modelos, como características para o sistema (tirando uma característica que surge do par modelo-sistema). A compreensão do sistema e suas características exigem, segundo os autores, os modelos UML e um código-fonte. São características do modelo:

- *Complexidade*: o esforço necessário para compreender o modelo ou sistema;
- *Rastreabilidade*: o quanto as relações entre as decisões de projeto estão explicitadas;
- *Modularidade*: o quanto as partes do sistema estão estruturadas e separadas de forma a serem compreendidas isoladamente;
- *Completude*: o sistema tem completude conforme as funcionalidades implementadas correspondem e cobrem todos os requisitos especificados. Um modelo possui completude quando suas partes que se sobrepõem em diferentes modelos contém os mesmos elementos e relações, com o sistema sendo completamente descrito pelo modelo;
- *Consistência*: a característica da existência ou não de informações conflitantes no modelo;
- *Comunicabilidade*: o quanto o sistema facilita a especificação de entradas e resulta em saídas de forma que seu conteúdo seja de fácil assimilação e úteis ao usuário;

- *Auto Descrição*: se o modelo ou sistema contém informação o suficiente para que um leitor consiga determinar os objetivos, suposições, regras, limites, relações, entradas, saídas, componentes e status;
- *Detalhamento*: o quanto o modelo descreve detalhes relevantes do sistema;
- *Equilíbrio*: o modelo possui equilíbrio quando todas as partes do sistema são descritas em um nível igual ou equiparável a todas as outras características do sistema. O sistema tem equilíbrio quando todas as suas partes estão iguais em relação a outras partes do sistema;
- *Concisão*: o quanto o modelo descreve o sistema de forma prática e direta sem ser desnecessariamente extenso;
- *Estética*: o quanto o layout gráfico do modelo é agradável e facilita o entendimento do sistema descrito; e
- *Correspondência*: característica do par modelo-sistema. A consistência existe quando os elementos, relações e decisões de projeto do sistema são as mesmas encontradas tanto no modelo quanto no sistema implementado.

O quarto e último nível apresenta métricas e regras para o modelo de qualidade desenvolvido por Lange e Chaudron (2005). As métricas são utilizadas para medir algo do domínio empírico para o numérico, enquanto as regras são mais um mapeamento binário ("Caso de Uso Y deve apresentar característica X"). As métricas são retiradas de outros autores, somente são apresentadas as pertinentes para esse modelo de qualidade.

Os autores utilizam como métricas (LANGE E CHAUDRON, 2005):

- *Dinamicidade*: mede a complexidade do comportamento interno de uma classe. Toma informações de diagramas de interação;
- *Razão*: a razão entre o número de elementos por classe (número de métodos por classe);
- *Profundidade da Árvore de Herança*: DIT – *Depth of Inheritance Tree*;
- *Acoplamento*: o número de relações que um elemento tem com outros elementos;
- *Coesão*: mede o quanto as partes de uma classe são necessárias para realizar uma única tarefa;
- *Complexidade de Classe*: mede o esforço necessário para entender uma classe;
- *Número de Classes por Caso de Uso*: NCU – *Number of classes per use case*;
- *Número de Casos de Uso por Classe*: NUC – *Number of use cases per class*;

- *Fan-In*: o número de associações que entram na classe. Mede o quanto outros elementos utilizam os serviços da classe;
- *Fan-Out*: o número de associações que saem da classe. Mede o quanto a classe usa serviços providenciados por outros elementos do modelo;
- *Convenções de Nomenclatura*: se o modelo segue uma convenção de nomenclatura para seus elementos;
- *Padrões de Projeto*: se o modelo segue padrões de projeto;
- *Número de Linhas Cruzadas em Um Diagrama*: NCL – *number of crossing lines in a diagram*;
- *Definições Múltiplas*: múltiplas definições de um elemento com o mesmo nome nos modelos;
- *Cobertura do Diagrama de Interação*: mede o quanto os diagramas de interação cobrem os elementos descritos nos diagramas estruturais;
- *Mensagem necessita de Métodos*: essa regra diz que cada mensagem em um diagrama de interação deve corresponder a um método definido no diagrama de classes;
- *Correspondência no Código*: mede o quanto o código corresponde aos modelos; e
- *Comentário*: mede o quanto o modelo tem comentários.

Por fim, mostram-se as relações entre os vários níveis do modelo na Figura 3.1. As flechas indicam a relação do nível mais baixo para o mais alto de abstração e qual característica corresponde a qual propósito, e qual propósito a qual uso do modelo.

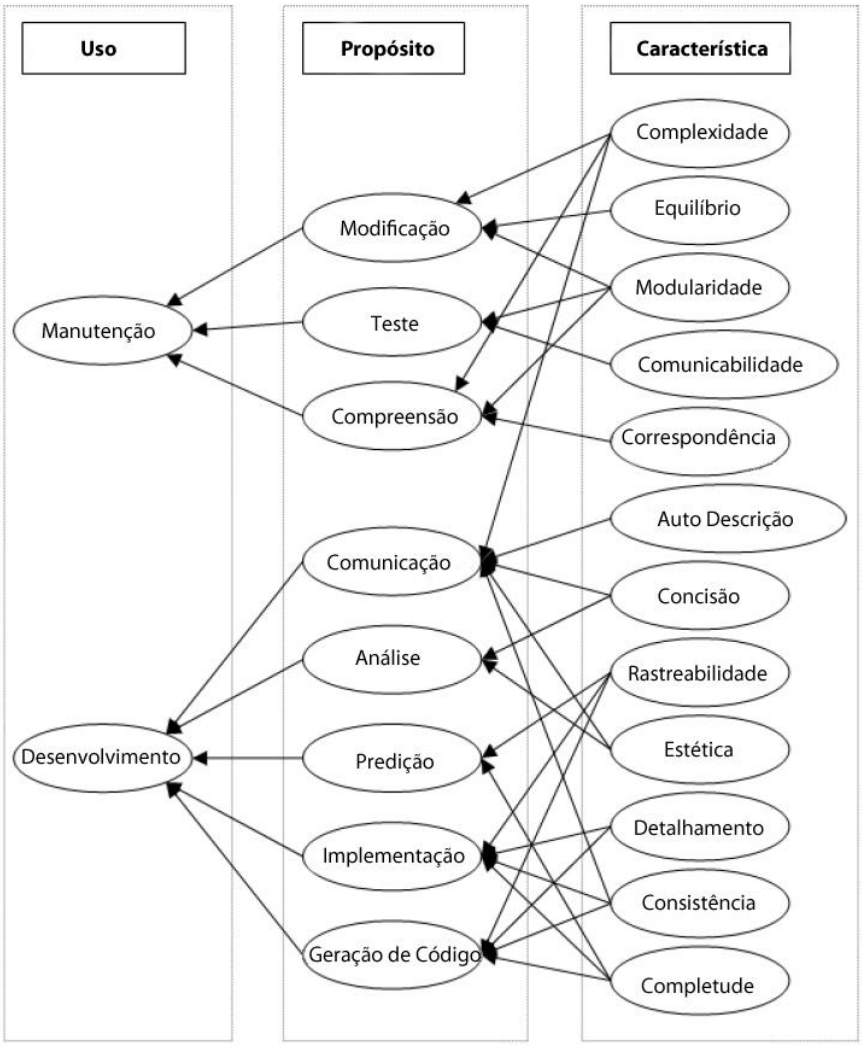

Figura 3.1 - Modelo de Qualidade proposto por Lange e Chaudron (2005). Fonte: idem.

O modelo dos autores foi pensado para modelar com qualidade para as abordagens orientadas a modelo tendo em mente as preocupações com esse processo já comentadas. Apesar de modelos terem surgido com a ideia de criar software com maior qualidade, nem sempre os modelos são bem feitos. Ao usar os passos dos autores, o processo de modelagem teria maior qualidade. É importante ressaltar também que os autores propõem um conjunto de métricas que é essencial para medir e avaliar a qualidade no modelo construído.

3.2.2 Seis objetivos de Mohagheghi, Dehlen e Neple

Em seu trabalho, Mohagheghi et al.(2009) fazem uma análise sobre as definições e abordagens de qualidade em modelos em um desenvolvimento orientado a modelos. Os autores fazem uma revisão das convenções existentes em modelagem de software e dos conceitos relacionados ao novo paradigma proposto. Em sua análise, os autores propõem que existem seis objetivos diferentes, em relação à qualidade em modelos. São eles:

- *Corretude*: o modelo contêm os elementos certos e as relações corretas entre eles. Também representa não violar regras e convenções assumidas no projeto;
- *Completude*: o modelo deve ter toda a informação necessária e relevante, sendo detalhada o suficiente de acordo com o propósito do modelo;
- *Consistência*: não deve haver contradições no modelo. A consistência deve existir entre visões ou diagramas que estão no mesmo nível de abstração e que representam os mesmos aspectos em níveis diferentes;
- *Compreensão*: deve ser compreensível para os usuários alvo do modelo, tanto humano como máquina;
- *Limitação*: o modelo está de acordo com os propósitos da modelagem e o tipo de sistema desenvolvido; e
- *Adaptabilidade*: o modelo deve ser de fácil mudança e adaptação, também é importante que ele possa ser mudado de forma a evoluir rápido e continuamente.

Os seis objetivos auxiliariam a melhorar a qualidade do modelo feito pelo desenvolvedor. Ao iniciar o desenvolvimento, deve-se assumir um conjunto de regras e convenções que serão usadas durante o desenvolvimento, por exemplo, uma convenção para nomenclatura dos elementos nos diagramas. Deve-se manter uma boa estrutura semântica e sintática nos modelos projetados. O modelo é semanticamente correto se as relações estão corretas e os diagramas representam de forma fiel aquilo que é o software a ser produzido. Assim mostra-se a importância da corretude para a abordagem MDE (MOHAGHEGHI et al., 2009).

Os modelos devem conter toda informação precisa em sua documentação e diagramas para que exista uma melhor interpretação pelas ferramentas. Os modelos completos podem ser usados para testes e para criar novas tarefas do projeto (MOHAGHEGHI et al., 2009). Como as ferramentas fazem interpretação dos diagramas para a transformação, é essencial que os modelos tenham uma informação completa, permitindo que exista uma interpretação pela máquina mais fiel àquilo que o desenvolvedor pensou ao desenvolver o projeto.

Um projeto de um software não é feito somente de um diagrama. Pelo mesmo motivo já citado, é importante que o modelo não tenha contradições. Essas contradições causam inconsistências no modelo o que podem acarretar problemas no desenvolvimento de acordo com a abordagem orientada a modelos (MOHAGHEGHI et al., 2009). Lange e Chaudron (2005) ressaltam o risco que existe de inconsistências em modelos com muitos diagramas. As inconsistências podem não só ocorrer entre diagramas do mesmo nível de abstração, como em níveis e modelos diferentes. A consistência é fundamental para a sua correta interpretação pela ferramenta e transformação do modelo.

A compreensão é importante de duas maneiras. Para os humanos é importante que os diagramas sejam legíveis, bem desenvolvidos esteticamente, usando uma linguagem de fácil compreensão ao usuário e que as interações sejam claras. Também é importante saber qual modelo é mais adequado para qual público. A segunda maneira em que a compreensão é importante é para a máquina, os modelos devem ser precisos de forma sintática e semântica para que ajude a precisão da transformação (MOHAGHEGHI et al., 2009).

O modelo deve ser uma coisa limitada a seu uso. Reconhecendo o uso e o tipo de sistema pode-se decidir quais diagramas são relevantes para a construção do projeto e qual o nível de abstração que se deve utilizar. Um modelo que está dentro de suas limitações não apresenta informações desnecessárias e não é mais complexo ou detalhado do que é preciso (MOHAGHEGHI et al., 2009). Existem várias motivações para manter o diagrama limitado, uma delas é aumentar a legibilidade do modelo, porém, é importante ressaltar que para transformações é ideal que o modelo seja detalhado a um nível executável.

Durante o desenvolvimento de software o domínio do problema e os requisitos do projeto mudam com frequência, isso torna necessário que os modelos e diagramas sejam modificados várias vezes para se adequar aos novos requisitos ou domínio. Os modelos devem ser desenvolvidos com isso em mente, de forma que as mudanças não causem grande

impacto e a estrutura do projeto dê suporte para modificações. As ferramentas e linguagens utilizadas também devem ser adequadas para esse aspecto (MOHAGHEGHI et al., 2009).

Os autores colocam que esses são os objetivos que se deve focar na construção de modelos para garantir qualidade nos mesmos. Eles nomeiam o modelo de objetivos 6C (*6C goals*) devido aos nomes dos objetivos em inglês. Cada objetivo se relaciona com um aspecto do desenvolvimento. Consistência ocorre internamente dentro do modelo. Compreensão ocorre entre os usuários (humanos ou ferramentas) e o modelo. Corretude é um objetivo entre a linguagem e as regras de modelagem e o projeto, também ocorre entre o ambiente e o modelo. Completude, Limitação e Adaptabilidade envolvem o ambiente e o projeto. Na Figura 3.2 pode-se observar essas relações.

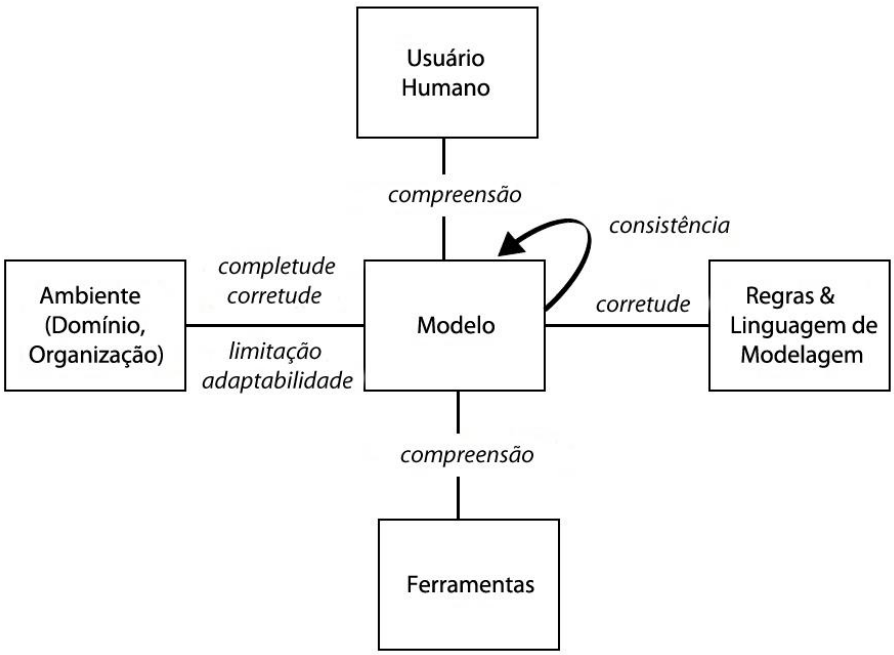

Figura 3.2 - Os objetivos do modelo de Mohagheghi et al. (2009). Fonte: idem.

Finalmente, os autores expõem que é importante lembrar que todos objetivos e propriedades discutidas dependem daquilo que é percebido pela pessoa que modela o sistema. O modelo deve ser uma representação do sistema e tem que ser completo em relação àquilo que o sistema representa. O modelo também é dependente do ambiente em que o sistema se encontra, isso é, depende do domínio e dos propósitos do modelo. Levando em conta os princípios de Lange e Chaudron (2005), deve-se ressaltar que a precisão desses objetivos

depende do desenvolvedor e do contexto em que o modelo se encontra e a partir do propósito, ou seja, se os modelos são utilizados para implementação, teste ou manutenção de sistemas.

Mohagheghi et al. (2009) concluem sua análise dizendo que outros objetivos discutidos na literatura podem ser alcançados através do modelo 6C. Os autores exemplificam colocando que um modelo que é correto, completo e consistente não permite múltiplas interpretações e todos esses objetivos são importantes para dar reusabilidade aos modelos.

O modelo 6C foi construído a partir de uma extensa análise da literatura feita pelos autores. Isso torna o modelo conciso e abrangente a maior parte das preocupações que existem em termos de qualidade em modelos. Os autores procuram melhorar a qualidade de desenvolvimento de modelos agrupando objetivos e processos encontrados na literatura. O modelo dos autores se torna prático e de fácil entendimento.

3.3 Qualidade em transformações

A qualidade em transformações é parte fundamental para garantia da mesma no processo de desenvolvimento em MDE. Como apresentado anteriormente, a qualidade de software em um ciclo de desenvolvimento tradicional, usualmente se foca na qualidade do código e dos processos que realmente impactam o produto final.

Isso quer dizer que a qualidade se preocupa muito com o desenvolvimento em si, partindo de práticas de programação, boas práticas para evitar erros de implementação, modelos iterativos e incrementais que melhoram o processo a cada passo, entre outros.

Em MDE, esse cenário é modificado. A parte de implementação e programação é toda feita por ferramentas e regras de transformações. Assim, muda-se a forma como se obtém e garante qualidade no desenvolvimento. Uma delas é melhorar a qualidade da produção de modelos e a outra é melhorar as transformações.

Como todos os novos conceitos e tecnologias estudados, MDE ainda está caminhando para se realizar a ideia como descrita. Então, as transformações ainda não são estabilizadas, padronizadas e totalmente confiáveis. Conforme se aumenta o grau de abstração dos modelos que devem ser transformados, mais difícil é manter a transformação fiel àquilo que o

desenvolvedor queria expressar (AMELLER E FRACH, 2009; KEPPLE, 2006; PÉREZ-MARTÍNEZ E SIERRA-ALONSO, 2006).

Mellor e Balcer (2002) ao criar a xUML e detalhar em seu livro, compartilham esse pensamento. Os autores teorizaram que a interpretação de modelos muito abstratos seria muito custosa e não traria o resultado esperado. Muitas lacunas ficariam faltando em modelos com pouca ou nenhuma informação para que elas fossem preenchidas automaticamente. Assim, eles propuseram tornar modelos executáveis. Isso é realizado na medida em que se insere mais informações nos modelos, diminuindo sua abstração, porém dando um poder maior para a transformação automática.

Pode-se fazer um paralelo entre os compiladores e as transformações. Os compiladores, como dito na Seção 2.1, também levaram desconfiança e faziam a interpretação de um código em linguagem mais próxima a do desenvolvedor para a linguagem máquina. As transformações de MDE são o mesmo processo com uma abstração maior, sendo assim, existe uma necessidade de certificar-se que as interpretações dos computadores ocorrem precisamente. Essa é a visão compartilhada por autores como Ameller e Frach (2009), Rech e Bunsen (2008) e Kepple (2006).

A transformação é um software, criado por um programador, que segue regras para interpretar um modelo e criar um novo modelo (transformações M2M) ou para código (M2T). Por ser um software, parte do processo de garantia de qualidade de software passa pelas práticas tradicionais da área. Usualmente, essas práticas passam por conceitos comuns descritos por Koscianski e Soares (2007), assim como as práticas descritas pelas normas ISO/IEC 9126 (2001), ISO/IEC 14598 (1999) e NBR ISO/IEC 9126-1 (2003).

Os outros estudos que tangenciam a qualidade em transformações usualmente referem-se a questões de reuso principalmente (JIN e GUISHENG, 2010), também se inclui pesquisas em campos de utilização e adaptação de padrões de segurança (SHIROMA et al., 2010), transformação e aplicação de softwares com requisitos não funcionais (STERRIT E CAHILL, 2008) e aplicações para sistemas embarcados (BOUKHANOUFA et al., 2010).

Os modelos e linguagens de transformações, mencionados anteriormente, também entram em qualidade de transformação. Os modelos e linguagens surgem da necessidade de linguagens específicas para modelar como as transformações devem ocorrer. Esse trabalho não se prende a uma linguagem ou modelo específico, por isso não serão discutidas as

particularidades de cada um, a documentação e explicação de cada uma das linguagens podem ser encontradas em ATL (2011) e VIATRA2 (2011).

Vanhooff et al. (2007) apresentam um processo de transformação que estimularia o reuso e seria independente de tecnologia. Os autores em seus estudos primeiros fazem uma análise do cenário de transformações. Vanhooff et al. (2007) identificam que as limitações das transformações são:

- Especificação incompleta;
- Especificação imprecisa;
- Presos a uma tecnologia; e
- Administração de megamodelos.

A especificação incompleta refere-se às linguagens e modelos de transformação não abrangerem todo o processo de transformação. Os autores colocam que existem modelos que contemplam transformações M2M, porém não M2T. Assim como existem aquelas que só dão suporte a metamodelos ou a esqueletos. Não existe uma ferramenta que consiga suportar todo o processo de desenvolvimento, fazendo com que as equipes tenham que usar várias ferramentas diferentes.

A especificação imprecisa que os autores se referem é o fato que nenhum método define exatamente as variações de modelo, delimitação e estrutura. A UML foi criada para acomodar uma variedade de modelos com diferentes níveis de abstração e especificações. Porém, os autores defendem que durante as transformações isso não é desejável, já que o código segue um conjunto de regras e interpreta os modelos conforme as definições desse conjunto, qualquer modelo que fuja dos padrões programados causa problemas para as ferramentas de transformação, mesmo que seja aceitável pela UML.

Essa constatação vai de acordo com a proposta de Mellor e Balcer (2002) que definem a xUML como uma forma mais delimitada e específica da UML para que as transformações ocorram de forma melhor. Os autores também defendem inserir mais informações no modelo, perdendo um pouco sua abstração, porém assegurando uma transformação mais confiável.

Para Vanhooff *et al.* (2007), um dos principais problemas dos processos de transformações é que eles são ligados a uma tecnologia específica, necessitando que os desenvolvedores tenham um conhecimento muito grande da tecnologia para conseguir desenvolver eficientemente usando MDE. O principal problema evidenciado pelos autores são

as diferenças entre as tecnologias que cada um dos modelos é relacionado. É muito difícil cruzar as tecnologias diferentes usadas nas transformações sem a interferência externa.

A administração de megamodelos é a forma com que os autores encontram de expressar que as implementações de MDE preocupam-se com as transformações individuais, não se preocupando com a corrente de transformações que um processo de desenvolvimento normal teria. A necessidade de administrar megamodelos entra em foco principalmente no caso de reuso de transformações.

As transformações são compostas de várias subtransformações que ocorrem. Muitas vezes para cada tipo de transformação é necessário elaborar uma nova regra e uma nova forma de transformar, assim, estimula-se a criação de regras mais genéricas para as diferentes necessidades, impulsionando a ideia de reuso.

A partir de sua análise, Vanhooff et al. (2007) propõem um modelo para representar as transformações chamado UTR (*Unified Transformation Representation*). Na Figura 3.3, o metamodelo dos autores pode ser observado.

A representação dos autores é separada em duas partes distintas: especificação e execução. A especificação é a parte que encapsula os elementos que envolvem a implementação do sistema em MDE. A execução é a instância de uma transformação. Na Figura 3.3 isso é visualizado através da linha que divide o modelo, o que está acima da linha é especificação e o que está abaixo faz parte da execução.

A classe *TFSpecification* é onde estão encapsuladas a implementação e a especificação externa do modelo. Em *AtomicTFSpecification* estão as características relacionadas a tecnologia de transformação escolhida. *CompositeTFSpecification* é a expressão da corrente de subtransformações que existe em uma instância, informação herdada por *TFChain*. Os detalhes específicos da implementação são colocados nas classes *ATLImpl*, *MTFImpl* e *JavaImpl*.

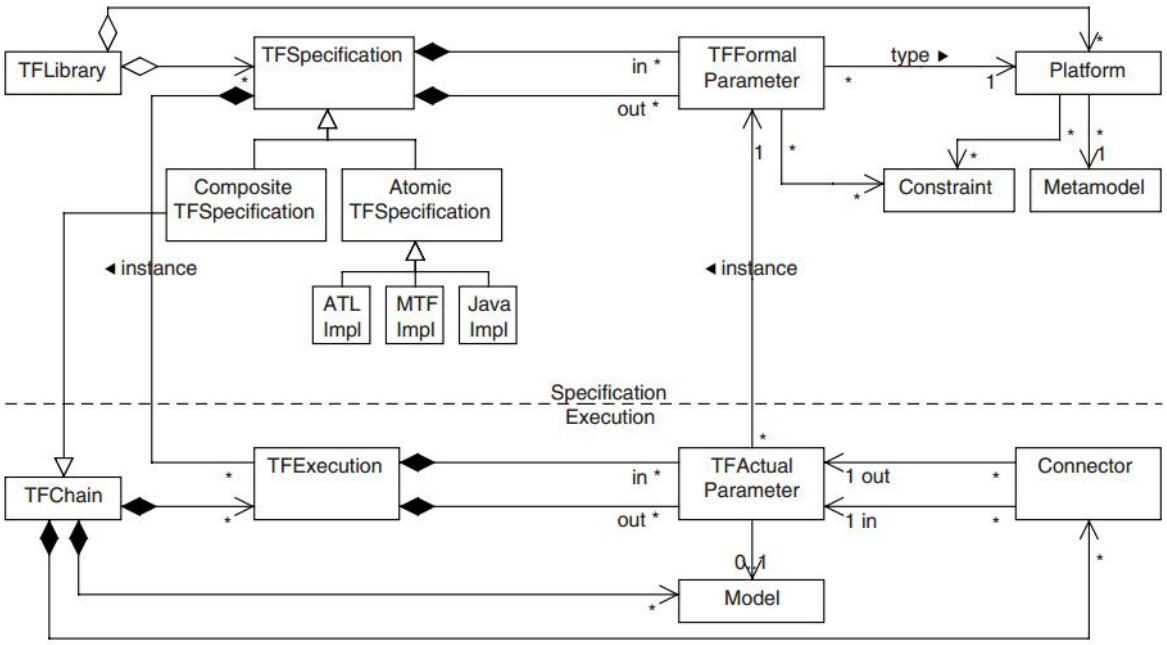

Figura 3.3 - Metamodelo de UTR. Fonte: Vanhooff et al. (2007)

A classe *TFSpecification* é onde os modelos de entrada e saída são guardados durante o processo. Os seus parâmetros são descritos em *TFFormalParameter* e os tipos de modelo em *ModelingPlataforms*. Essas duas classes são o que definem e delimitam os modelos encontrados naquela classe, podendo ser agrupadas em *TFLibrary*.

A parte de baixo da figura representa a execução. *TFExecution* e *TFActualParameter* são as instâncias de execução das classes *TFSpecification* e *TFFormalParameter*. A classe *TFActualParameter* é composta pelo modelo (representado por *Model*) concreto. A classe *Connector* é utilizada para fazer a comunicação entre *TFExecution* e *TFActualParameter*. Através dele podem-se verificar modelos, pausar execução e outras atividades durante a transformação. Uma conexão só é válida se as tecnologias e os modelos forem compatíveis. A classe *TFChain* é que representa a corrente de subtransformações em instância de execução.

Por fim, Vanhooff *et al.* (2007) definem três papéis diferentes no processo de transformação. São eles:

- *Desenvolvedor de Transformação* – é a pessoa que implementa a transformação usando a tecnologia adequada ou escolhida;

- *Especificador de Transformação* – é o responsável pela especificação externa da transformação. Pode ser especificador de duas formas, uma sendo um projetista da transformação, reunindo requisitos e definindo as especificações. Ele também é um coletor, no qual ele deve procurar por transformações que já existam que resolvam os requisitos do sistema atual; e
- *Montador de Transformação* – é a pessoa que seleciona a especificação apropriada e compõe a execução de uma corrente de transformações para atingir um objetivo maior da transformação.

Os papéis que os autores propõem cobrem todas as responsabilidades que existem dentro do processo de criação de transformações. O processo proposto nesse estudo usa noções e se baseia em conceitos do modelo de Vanhooff et al. (2007).

Dentro dos modelos de qualidade, o estudo é o que se apresenta mais compatível, fazendo definição de papéis e procurando criar processos independentes de tecnologia. É importante ressaltar que é necessário realizar validação e controle dos modelos de transformação.

3.4 Validação e controle

Validação e controle em MDE ainda é um tópico de discussão. Assim como a própria qualidade e o processo de MDE como um todo, a validação e controle podem ser classificados em duas partes. Uma delas se volta a validar e mensurar os modelos construídos. Outra se volta a certificar-se de que o processo de criação e transformações está ocorrendo com qualidade.

A mera aplicação de um modelo de qualidade a um processo qualquer não implica que o mesmo resultará em um produto de qualidade ou que o processo tenha sua qualidade garantida. Assim, é necessário a verificação e controle para garantir que o processo possa ter qualidade e resultar num bom produto. A partir dos números e resultados dos testes e validações feitas, os profissionais e desenvolvedores podem saber o que deve ser ajustado ou melhorado.

Nessa subseção são discutidas as técnicas de validação e controle em MDE baseado nos métodos apresentados durante essa seção. Para melhor entendimento, serão discutidas a validação e controle de modelos e transformações separadamente.

3.4.1 Validação de modelos

A validação em termos de modelo representa a utilização de métricas e métodos de mensurar modelos para validar sua construção. O controle do processo é importante para que os desenvolvedores obtenham uma retroalimentação do seu trabalho e encontrem erros e partes que foram construídas pobremente.

Existem na literatura vários trabalhos que procuram construir modelos, medidas e métricas para modelos. Genero et al. (2004) demonstram métricas para modelos UML, os mesmos autores em 2005 procuram detalhar métricas para diagramas de classe (GENERO et al., 2005). Genero et al. (2002) definem métricas para modelos dinâmicos. Outros autores como Mahmood e Lai (2005) se preocupam com modelos de componentes e como medi-los. Ma et al. (2004) procuram como aplicar métricas de OO para os modelos UML, eles evoluem o trabalho em Ma et al. (2005) procurando uma avaliação melhor dos modelos.

Ainda na validação de modelos pode-se destacar o trabalho de Berenbach e Barotto (2006) que procuram listar métricas e boas práticas para o desenvolvimento dos requisitos em MDE. Os autores colocam nos seus objetivos as preocupações mais com o processo de desenvolvimento do que nos modelos em si.

Apesar de existirem vários estudos de métricas em modelos e sua aplicação, não existe uma forma definitiva de aplicação. O uso das métricas depende do contexto, do sistema a ser construído e das tecnologias escolhidas pelos desenvolvedores para modelar e transformar seu sistema.

Lange e Chaudron (2005), trabalho discutido anteriormente, são autores que sugerem e definem métricas e seus usos dentro de seus modelos. Porém esse é um caso raro. Usualmente os autores não definem e deixam aberto à escolha dos desenvolvedores. O principal obstáculo entre métricas de modelos e o desenvolvimento MDE é a implicação que certas medidas têm

no produto final não fica claro. É difícil perceber em medidas como o número de linhas cruzadas terá impacto no produto final.

Monperrus et al. (2008) se baseiam nesse fato em sua pesquisa. Os autores dizem que existem três perguntas com que os especialistas em métricas devem se preocupar. São elas:

- Quais métricas se relacionam a funcionalidade?
- Quais métricas se relacionam com a estabilidade e precisão?
- Quais métricas que podem ser relacionadas à manutenibilidade?

Os autores colocam que várias vezes as métricas são obscuras sobre seu real impacto no desenvolvimento, isso se existir algum. Assim, é necessário primeiro compreender isso antes que o desenvolvedor possa decidir as métricas para seu processo.

Seguindo o modelo de Lange e Chaudron (2005) e aliando com os conceitos de Monperrus et al. (2008) pode-se dizer que é importante que primeiramente os desenvolvedores tenham conhecimento e avaliem os propósitos de seu sistema e dos modelos construídos, para então decidir quais métricas que devem ser utilizadas durante o processo.

Outro ponto de destaque para o controle e qualidade de modelos é a refatoração. A refatoração consiste em analisar os modelos e alterar eles em sua estrutura interna, sem que se mude o comportamento geral do sistema. A refatoração tem como objetivo criar modelos mais claros e eficientes, que mantenham todas as funcionalidades dos modelos construídos anteriormente (MENS et al., 2008).

O refatoramento de modelos é estudado e descrito por autores como Mens et al. (2008), Hosseini e Azgomi (2008), Barbosa e Meng (2008) e France et al. (2003). Conforme colocam Barbosa e Meng (2008) a refatoração faz parte da construção de uma estrutura mais estável para o desenvolvimento em MDE.

3.4.2 Validação de transformações

A validação de transformações é um processo mais tradicional. As transformações podem ser vistas como softwares comuns. Sendo assim, sua avaliação leva em conta aspectos comuns de avaliação descritos na literatura.

Isso pode ser visto em trabalhos como o de Rahimi e Lano (2011) que utilizam o método GQM (*Goal Question Metric*) para avaliar diferentes métodos de transformação. Eles também introduzem uma forma de medição orientada aos objetivos da transformação.

As transformações usualmente devem seguir modelos de qualidade como ISO/IEC 9126 (2001) e ISO/IEC 14598 (1999). Sua avaliação também segue os mesmos princípios. A avaliação de software deve ser feita através de testes e práticas descritas nas duas normas, os itens que a ISO/IEC 14598 (1999) sugere que sejam avaliados são:

- Funcionalidade;
- Adequação;
- Acurácia;
- Interoperabilidade;
- Conformidade;
- Segurança de acesso;
- Confiabilidade;
- Maturidade;
- Tolerância a falhas;
- Recuperabilidade;
- Usabilidade;
- Inteligibilidade;
- Apreensibilidade;
- Operacionalidade;
- Eficiência;
- Tempo (de resposta e execução);
- Recursos;
- Manutenibilidade;

- Analisabilidade;
- Modificabilidade;
- Testabilidade;
- Portabilidade;
- Adaptabilidade;
- Capacidade para ser instalado;
- Conformidade; e
- Capacidade para substituir.

Esses são os aspectos que o software deve ter e podem ser avaliados. O método de testes também é descrito pelas normas e pela literatura. Conforme Koscianski e Soares (2007) o processo deve estar ligado por um planejamento do teste, reconhecendo quais características e especificações do software de transformação que serão analisados e qual o propósito do teste.

Os profissionais, então, podem planejar e decidir quais os melhores métodos para testes. Após isso, esses são executados e os seus resultados documentados. Por fim, fica a cargo dos analistas e testadores criarem relatórios sobre os resultados dos testes, incluindo quais partes são falhas e quais tem um desempenho que pode ser melhorado.

Rahimi e Lano (2011) colocam que algumas propriedades influentes às transformações de modelo devem ser levadas em conta quando se trabalha com sua validação. São essas propriedades:

- *Nível de abstração* – o grau de abstração refere-se às especificações que abstraem detalhes de implementação;
- *Modularidade* – como as especificações são organizadas e estruturadas dentro de si;
- *Compreensibilidade* – quão fácil é entender e analisar a especificação e a implementação;
- *Usabilidade* – o esforço que é requerido pelo desenvolvedor para construir e analisar a especificação; e
- *Implementação* – se existem maneiras eficientes de implementar as transformações.

Para os autores essas cinco propriedades devem ser observadas quando a equipe decide quais métodos de avaliação e medição serão utilizados durante o desenvolvimento. Também é destacado pelos autores que apesar dos altos níveis de abstração serem mais

concisos e mais próximos dos requisitos descritos (tornando sua validação mais fácil), menos eficiente será a implementação.

A principal contribuição dos autores é validar o método GQM como uma abordagem útil a definição de métricas para transformações em MDE. Eles descrevem que os objetivos ajudam a identificar, analisar e interpretar dados relevantes do sistema dentro do *framework*. O método GQM então se torna um método chave para identificação das métricas e testes relevantes para a transformação.

3.5 Considerações

Nessa Seção procurou-se aprofundar nos estudos dentro de qualidade em MDE. Os estudos apresentados aqui se relacionam a essa pesquisa de forma norteadora e de base para o processo proposto.

Antes de poder adaptar e criar um novo processo integrado, é preciso compreender como o processo de MDE ocorre e quais são as características inerentes dos estudos de qualidade. A divisão aqui proposta é a que mais se adequa aos estudos encontrados. Ao dividir em qualidade de modelos, transformações e validação, se identificam as partes que compõem o todo.

Isso é evidente no desenvolvimento em MDE. A criação de novas regras de transformação e a sua reutilização não ocorrem durante o desenvolvimento, tanto porque isso seria contra produtivo. Assim fica claro que existe uma separação na construção de transformações e dos modelos (ou do sistema em si). Mesmo assim, também é evidente que um está intrinsecamente ligado ao outro.

As características e especificações dos modelos escolhidos para fazer o desenvolvimento devem guiar a transformação para o caminho correto. Os modelos também devem se guiar pelas limitações e possibilidades que são oferecidas pelas suas transformações.

Durante a pesquisa sobre qualidade em MDE, identificou-se que existem alguns estudos que se destacam, e por isso foram escolhidos como base. Lange e Chaudron (2005)

apresentam um modelo de construção de modelos com qualidade com a intenção de modelos criados voltados a MDE. Esse estudo é o que influenciou os estudos posteriores de qualidade em modelos nos domínios específicos. Ainda em qualidade em modelos, Mohagheghi *et al.* (2009) apresentam seis objetivos que devem ser observados na construção de modelos durante o desenvolvimento em MDE que agrupam e sintetizam quase uma década de desenvolvimentos na área.

Vanhooff *et al.* (2007) definem uma arquitetura para transformações genéricas e conseguem descrever o processo de transformações em MDE, além de definirem os papéis atrelados ao desenvolvimento de transformações. Por fim, utilizou-se de base vários estudos para embasar o conhecimento sobre métricas e controle de modelos e transformações.

Assim, os autores destacados são aqueles em que esse estudo se baseia e similaridades podem ser vistas no processo nas próximas seções. É importante ressaltar que alguns desses trabalhos são vistos como trabalhos correlatos a esse, conforme será discutido no próximo capítulo.

4 TRABALHOS CORRELATOS

Ao longo da última década, vários estudos em MDE foram realizados. Nesse capítulo são apresentados os trabalhos que se relacionam ao tema desta dissertação. Isso tem como objetivo demonstrar trabalhos em que os aspectos têm ligações com o que é feito nessa dissertação. No capítulo anterior foi discutida a qualidade em MDE, no qual se dividiu os estudos da área em MDE. Os trabalhos relacionados a essa dissertação também se classificam no mesmo aspecto.

Aqui se apresentam os estudos que tem elementos que se relacionam de forma mais próxima a esse trabalho. Não foram encontrados estudos, até o momento de elaboração dessa dissertação, que se relacionem na forma de descrição de um processo integrado.

Lange e Chaudron (2005) consideram que a evolução dos conceitos de MDE e o avanço da popularidade da abordagem faz com que se necessite de métodos estabelecidos para garantir e medir qualidade em modelos. Em seu trabalho, os autores propõem um modelo de qualidade centrado em modelos. Os autores também retiram da literatura várias métricas e as relacionam para usar em seu modelo e relatam o que cada uma avaliaria. O trabalho é influente no tema qualidade de modelos para MDE. É importante notar que os autores não fazem considerações sobre o processo de desenvolvimento, somente sobre o modelo de qualidade.

Vanhooff et al. (2007) enxergam as necessidades de se definir processos mais genéricos para desenvolvimento de software utilizando o novo paradigma. Sendo assim, os autores propõem em seu trabalho um processo de desenvolvimento e criação de transformações. Segundo os autores, existe uma ordem e certas atividades que quando realizadas nessa ordem, podem garantir que o processo de transformação tenha qualidade.

Sua maior correlação com esse trabalho é o método utilizado. No estudo, os autores também procuram dar uma definição de processo a elementos de MDE.

Hutchinson, Rouncefield e Whittle (2011) realizaram um estudo que avalia a utilização de MDE na indústria. Nesse estudo, os autores confirmaram que o processo de

desenvolvimento em MDE é específico e não integrado. Foi constatado que, apesar de haver interesse, pequenas e médias empresas não aplicam MDE devido aos custos atrelados ao processo. Por não existir uma especificação genérica e integrada, resulta na situação em que somente grandes empresas têm recursos (humanos, financeiros, temporais, etc.) para investir em processos de MDE. Isso corrobora com a definição do problema de pesquisa desta dissertação.

Rahimi e Lano (2011) fazem um estudo sobre a validação e o desenvolvimento de transformações. As preocupações dos autores estão em avaliar o processo de construção de novas transformações. Em seu trabalho, os autores contribuem com a área introduzindo o controle de processos de desenvolvimento de transformações tendo objetivos MDE em mente.

Em pesquisa realizada, Berenbach e Barotto (2006) procuram criar formas de medir e avaliar modelos para os processos de *model-driven engineering*. Os autores se preocupam, principalmente, com o processo de desenvolvimento de modelos, apesar de também demonstrarem métodos e técnicas para os modelos em si.

Por fim, pode-se citar o trabalho de Mohagheghi *et al.*(2009). Neste trabalho, os autores procuram fazer uma revisão de toda a literatura de qualidade no desenvolvimento utilizando abordagens MDE. Os autores integram conceitos e sintetizando-os em formas mais práticas em relação a desenvolvimento de modelos com qualidade.

Na Tabela 4.1 pode ser visualizado um comparativo entre os trabalhos correlatos e os assuntos relacionados à MDE.

Tabela 4.1 - Comparação de trabalhos correlatos.

Trabalho	Qualidade em Modelos	Verificação de Modelos	Qualidade em Transformações	Verificação de Transformações	Integração do Processo de MDE
Lange e Chaudron (2005)	Sim	Sim	Não	Não	Não
Vanhooff *et*	Não	Não	Sim	Sim	Não

***al.* (2007)**					
Rahimi e Lano (2011)	Não	Não	Não	Sim	Não
Berenbach e Barotto (2006)	Não	Sim	Não	Não	Não
Mohagheghi *et al.* (2009)	Sim	Sim	Não	Não	Não
Esta dissertação	Sim	Sim	Sim	Sim	Sim

Esse trabalho se diferencia dos outros, pois propõe um *framework* que integra os aspectos estudados pelos outros autores em uma visão holística do processo de desenvolvimento em MDE. Além disso, o trabalho procura fazer utilização de práticas e modelos já consolidados para o desenvolvimento em MDE.

5 MÉTODOS

Este capítulo tem como objetivo definir o tipo de pesquisa deste trabalho, o seu objeto de estudo e descrever os procedimentos metodológicos. Busca-se, assim, dar um melhor entendimento da realização e construção do trabalho (YIN, 2001; JUNG, 2004).

5.1 Caracterização da pesquisa

Yin (2001) e Jung (2004) colocam que uma pesquisa pode ser classificada em termos de natureza, estratégia, tipo de pesquisa, área do conhecimento, tipo de desenho de pesquisa, método de pesquisa e local de desenvolvimento. Sendo assim, na Tabela 5.1 é apresentada a classificação nesses campos deste trabalho.

Tabela 5.1 - Classificação nos campos de pesquisa deste trabalho.

Natureza	Aplicada
Estratégia	Qualitativa
Tipo de pesquisa	Exploratória
Área do conhecimento	Ciências da Tecnologia
Tipo de Desenho de Pesquisa	Experimental
Método de Pesquisa	Descrição de aplicações
Local de Desenvolvimento	Laboratório

A natureza é classificada como aplicada, pois, conforme Jung (2004), a pesquisa aplicada ou tecnológica visa aplicar conhecimentos obtidos na geração de novos produtos e processos dentro de uma área. A estratégia qualitativa é utilizada quando não se consegue quantificar os resultados. Além disso, a pesquisa qualitativa é interpretativa e apresenta uma abordagem mais profunda aos dados (WOHLIN et al., 2000).

A pesquisa exploratória se define por uma pesquisa onde um assunto novo é estudado e relatado (YIN, 2001). Apesar de MDE não ser um assunto novo e pouco relatado, processos que integrem suas partes são áreas pouco relatadas, além de ainda existirem estudos em qualidade de MDE, logo se classifica essa pesquisa como exploratória.

A pesquisa tem caráter experimental, pois trabalha diretamente com a manipulação do processo. Segundo Jung (2004), as pesquisas experimentais são aquelas em que existe influência direta do pesquisador sobre seu objeto de estudo. O método de pesquisa utilizado foi descrição de aplicações. A descrição de aplicações pode ser conceituada como a descrição de um processo ou objeto de forma a ter como resultado uma lista de recomendações, ajustes ou sistemas similares que visam melhorar a aplicação descrita (JUNG, 2004; YIN, 2001; WOHLAN et al., 2000).

Na classificação de Dodig-Crnkovic (2002) pode-se citar esse trabalho como um trabalho de modelagem. Isto é, se estuda um conceito, processos ou modelos, e se criam novas abordagens ou meios de realizar o que é pretendido.

Por fim, o local de desenvolvimento é classificado como laboratório. Jung (2004) descreve as pesquisas que tem objetos, indivíduos e variáveis controladas pelos pesquisados no momento de pesquisa como sendo de laboratório. Isso não significa que a pesquisa tenha sido feita ou não em laboratório, porém que seus resultados vêm de ambientes controlados.

É importante ressaltar também que vários dos conceitos e dados utilizados na pesquisa foram obtidos como dados secundários. Isto é, os dados foram analisados por fontes terceiras e utilizados nesse estudo (YIN, 2001).

5.2 Objeto de estudo

O objeto de estudo dessa pesquisa é o processo de desenvolvimento em MDE e as práticas que trazem qualidade ao processo. Pode-se dizer que o estudo se concentra no processo que é apresentado no Capítulo 6 e nos conceitos apresentados anteriormente.

5.3 Procedimentos metodológicos

O primeiro passo do trabalho foi compreender o que é *Model-Driven Engineering* e qual era seu estado-da-arte. Sendo assim, procurou-se revisar artigos e livros sobre o assunto que se encontram na literatura específica. Durante a procura também se encontrou sites e fóruns que apresentavam desenvolvimento e conhecimentos sobre o assunto.

Ao identificar o estado-da-arte de MDE, que é a pesquisa e estudo sobre os processos e práticas envolvendo engenharia de software para aplicação em MDE, decidiu-se por investigar as questões envolvendo qualidade. Novamente foi feito uma revisão da literatura sobre o assunto, buscando entender e delimitar um problema de pesquisa. Durante a revisão literária, já se começou a construção e entendimento das divisões entre os aspectos de qualidade e quais as dimensões que ela apresenta em MDE. Por fim, identificou-se a necessidade de um processo integrado, delimitando esse então como problema de pesquisa.

Uma vez identificado o problema de pesquisa, buscou-se organizar o conhecimento da literatura e estruturá-lo de forma que fosse mais compreensível. Durante essa organização, investigou-se quais eram as práticas mais influentes e mais importantes de MDE, as quais deveriam ser levadas ao novo processo.

Durante o processo de desenvolvimento da pesquisa, foi necessário decidir uma ferramenta de descrição e construção de processos. Assim, encontraram-se duas soluções mais adequadas como ferramenta para este trabalho, das quais se decidiu utilizar a ferramenta *Eclipse Process Framework* (EPF, 2011). O principal motivo para essa escolha foi que a ferramenta é de uso aberto e tem uma comunidade online ativa e aberta que auxilia os usuários e estimula a cooperação múltipla das pessoas que optam por utilizá-la.

A partir do EPF e dos conhecimentos obtidos, construiu-se uma versão preliminar do processo pretendido. O objetivo foi identificar as atividades, tarefas, papéis e artefatos que estão atrelados ao desenvolvimento de MDE. Como base para o processo foi utilizado o processo unificado (RUP, descrito por Shuja e Krebs, 2008) e alguns de seus conceitos.

Após a primeira construção do processo, esse foi submetido a uma avaliação, submetendo o *framework* na forma de um artigo a uma comissão avaliadora. Nessa avaliação identificou-se que existiam práticas não descritas e outras informações que eram

desnecessárias, assim refinou-se o processo em uma nova versão. Novamente, repetiu-se o processo de avaliação, onde mais uma vez foi identificado que o processo poderia sofrer algumas mudanças. Por fim, reajustou-se o processo novamente, dividindo ele em suas partes e reestruturando suas atividades e papéis.

Para esta dissertação, procurou-se avaliar o *framework* construído de forma teórica e conceitual. Para isso, foi utilizado o método apresentado por Ericsoon et al. (2010) ao analisar outros *frameworks*. Também buscou-se estudos de caso documentados para ilustrar e comparar como seria utilizado o processo de MDE baseado no *framework* proposto.

5.4 Considerações

O objetivo deste capítulo foi caracterizar o presente trabalho e demonstrar quais processos metodológicos foram utilizados e qual foi o andamento da pesquisa. Também buscou-se demonstrar as ferramentas utilizadas e como foram escolhidos os autores que serviram de base ao estudo.

A compreensão do processo metodológico é fundamental para compreender os resultados da pesquisa e o que foi realmente realizado dentro daquilo que foi definido. A delimitação do objeto de estudo permite com que se saiba exatamente o que foi trabalhado dentro da pesquisa (JUNG, 2004; YIN, 2001).

No próximo capítulo é apresentado o processo que foi construído durante a pesquisa. Ou seja, o resultado da aplicação dos procedimentos metodológicos se instancia a seguir.

6 *FRAMEWORK* INTEGRADO PARA MDE

Neste capítulo é apresentado o processo proposto para integrar os aspectos de qualidade em MDE. O processo utiliza como base os *frameworks Rational Unified Process* e o *Open Unified Process* (OpenUP), sendo que os dois foram criados para descrever o processo de desenvolvimento de software, utilizando-se da visão iterativa e incremental. Neste trabalho, são propostas mudanças ao fluxo e em algumas atividades, visando englobar o processo de desenvolvimento em MDE. Um dos focos desses *frameworks*, e também do aqui proposto, é garantir que a qualidade seja inserida durante o processo de desenvolvimento.

Primeiramente, são apresentados os conceitos básicos para compreensão do processo. Depois serão discutidos quais papéis que surgem e se modificam com o novo paradigma. Por fim, se demonstra quais atividades, tarefas e artefatos são trabalhados durante o processo. Ressalta-se que aqui são demonstradas somente as mudanças pertinentes do processo, portanto os aspectos que não forem descritos são os que não tem mudanças significativas nas definições encontradas no RUP (SHUJA e KREBS, 2008) ou OpenUP (ECLIPSE, 2011).

6.1 Conceitos básicos

Ao ser definido ou adaptado, um processo contém vários elementos. Esses elementos são as partes que se interconectam e interagem entre si para representar e descrever o processo. Os elementos para definição de processos de software, usualmente, são oriundos da definição SPEM (*Software & Systems Process Engineering Metamodel Specification*; 2008) e daquelas encontradas nos *frameworks*. Na Figura 6.1 estão listados os elementos básicos do processo.

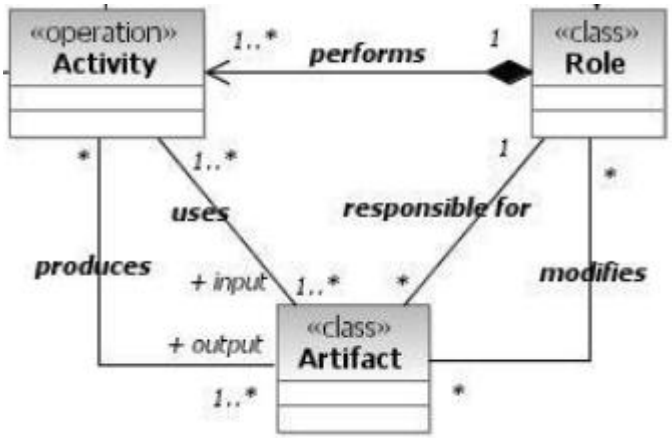

Figura 6.1 - Elementos Básicos do processo. Fonte: Bencomo (2005)

O processo é o que agrupa esses elementos, dando estrutura e sequenciamento às informações. Isso tudo ocorre dentro de uma dimensão temporal, os aspectos temporais do processo são apresentados em fases, que formam o ciclo de vida do projeto. Na tabela 6.1 estão descritas as fases do processo.

Tabela 6.1 - Fases do ciclo de vida do processo. Fonte: Shuja e Krebs, 2008.

Fase	Descrição
Concepção	Define-se o escopo e viabilidade do projeto
Elaboração	Projeta-se o software
Construção	Desenvolve-se o software
Transição	Faz-se a entrega do software

É importante notar que tradicionalmente, a elaboração trabalha mais com a construção da arquitetura do sistema em termos de modelos, enquanto a construção trabalha mais com a implementação em código. No processo de MDE isso é diferente. Define-se que a elaboração trabalhe com a projeção mais abstrata. Antes que os desenvolvedores possam construir a arquitetura, eles precisam compreender e criar modelos abstratos do software, os quais poderão ser de difícil interpretação para as ferramentas. Assim, a construção e detalhamento da arquitetura de software para realização das transformações devem ocorrer na fase de Construção.

Outro ponto que modifica são as disciplinas. O processo unificado define nove disciplinas diferentes no desenvolvimento de software. Enquanto as disciplinas são conceitos básicos e estáveis, existem algumas que mudam significantemente. Na Tabela 6.2 estão listadas as disciplinas do processo.

Tabela 6.2 - Disciplinas do processo unificado. Fonte: Shuja e Krebs, 2008.

Disciplina	**Descrição**
Modelagem de Negócio	Entender a estrutura e dinâmica do domínio do negócio
Requisitos	Elicitar os requisitos de software e delimitar o sistema
Análise e *Design*	**Projetar e criar a arquitetura do sistema**
Implementação	**Construir e testar os componentes do sistema**
Teste	**Avaliar o produto**
Implantação	Entregar o produto
Configuração e Mudança	Controlar os artefatos e suas mudanças
Gerenciamento de Projeto	Controlar e direcionar o projeto do sistema
Ambiente	**Criar estruturas para dar suporte ao desenvolvimento**

Os processos exibidos em negrito são os que apresentam uma mudança significativa que são descritos nesta dissertação. A Implementação é uma disciplina que prevê a construção de código. Sendo assim, essa é uma disciplina que incluiria o processo de criação e gerenciamento de transformações. Em sua descrição, porém, não seria incluso o desenvolvimento do sistema em si, pois esse passa por uma ferramenta de auto-codificação.

Logo, substitui-se a disciplina de Implementação. Ela deixa a existir para dar lugar a disciplina de Gerenciamento de Transformações. Ela passa a encapsular as atividades que guiam o desenvolvimento e adaptação de transformações dentro do ambiente de MDE.

Análise e *Design* passa a ser a principal disciplina de desenvolvimento. É ela que trabalha com o artefato central de desenvolvimento em MDE. A elaboração e definição da arquitetura do projeto são realizados e especificados de forma a serem desenvolvidos pelas ferramentas de transformação. Sua mudança mais importante está na preocupação maior em construir modelos e a criação de arquiteturas mais específicas.

Em Teste, ainda se prevê uma avaliação do produto do sistema. Ao trabalhar-se com uma abordagem orientada a modelos, os modelos também viram alvos de avaliação e

validação. Portanto, é necessário que existam meios e atividades que avaliem os modelos produzidos.

A mudança da disciplina de ambiente está em como adaptar o processo de software e os cuidados com as ferramentas a serem utilizadas para a transformação. É nessa disciplina que são realizados os processos adaptativos e básicos para utilização de ferramentas de transformação.

Ao analisar o processo de MDE, percebe-se que duas partes do processo de desenvolvimento em MDE podem ter classificações diferentes. Enquanto a construção de transformações é algo que se encaixa em um processo de desenvolvimento comum, a construção do sistema orientada a modelos entra em contradição com as definições de desenvolvimento comum de software.

De um lado, o desenvolvimento do sistema passa a ser baseado em modelos. Sendo assim, a implementação é substituída por uma tecnologia de transformação de modelos em código (ou uma corrente de transformações de modelos até chegar ao código). As partes que envolvem programação, neste caso, perdem sentido. A arquitetura do sistema, representada por modelos e pela documentação, passa a ser o artefato principal de desenvolvimento. Esse artefato é retirado dos requisitos do sistema.

De outro lado, existe a transformação. A transformação é um software (ferramenta) que através de regras nele implementadas, transforma modelo em outro modelo ou código. As transformações não englobam todas as relações e situações existentes em modelos e documentação. Sendo assim, regras de transformação podem ser adaptadas, inseridas, melhoradas ou criadas para integrar na ferramenta de transformação. Essas regras são geradas a partir dos requisitos que os arquitetos e *designers* de software encontram no desenvolvimento da arquitetura.

A intenção da utilização de MDE é conseguir fazer o reuso de regras de transformação e criar uma transformação global que, eventualmente, diminua os tempos de desenvolvimento através da automatização do processo de programação. Portanto, o desenvolvimento da transformação e sua manutenção é uma atividade constante em MDE. Ele é uma atividade que não se limita a um projeto ou a um ciclo de desenvolvimento. O mesmo, porém, tem um relacionamento forte com todos os projetos e ciclos de desenvolvimento.

Nota-se, portanto, que enquanto transformação e desenvolvimento do sistema ocorrem paralelos, eles não ocorrem no mesmo ciclo. Ainda assim, existe uma relação muito forte entre as duas partes. A transformação precisa dos requisitos que veem da necessidade dos desenvolvedores e dos modelos construídos por eles. Os desenvolvedores e o projeto do sistema necessitam do produto resultante da criação de transformações para que sejam transformados no produto final. O fato é que os dois processos não devem ocorrer no mesmo fluxo, porém não ocorrem sem os resultados do outro.

Logo, o processo de desenvolvimento se divide em duas partes. Uma é representada pela disciplina proposta nesse trabalho, chamada de gerenciamento de transformações A outra parte do processo, é a criação e estruturação do sistema em alto nível de abstração e a transformação dos modelos para código. A seguir são definidas as mudanças nessas disciplinas.

6.2 Análise e *Design*

Dentro do processo essa disciplina abrange duas fases do ciclo de desenvolvimento de software. Na fase de concepção, a disciplina deve estabelecer se o sistema é viável Em elaboração, a disciplina se foca primeiramente em avaliar as soluções que podem ser utilizadas. Depois tem objetivo de criar uma arquitetura e dar início às análises de sistema.

Observa-se que, dependendo do projeto, as modelagens mais abstratas são feitas primeiramente, em um segundo momento a arquitetura do sistema é especificada e refinada através da documentação e modelos mais específicos. A elaboração tem como objetivo trabalhar totalmente o artefato principal de desenvolvimento para a transformação.

Na Figura 6.2 é apresentado o diagrama de atividades para a disciplina de Análise e Design. Nota-se que a disciplina tem somente uma atividade dentro da fase de concepção, sendo essa atividade opcional.

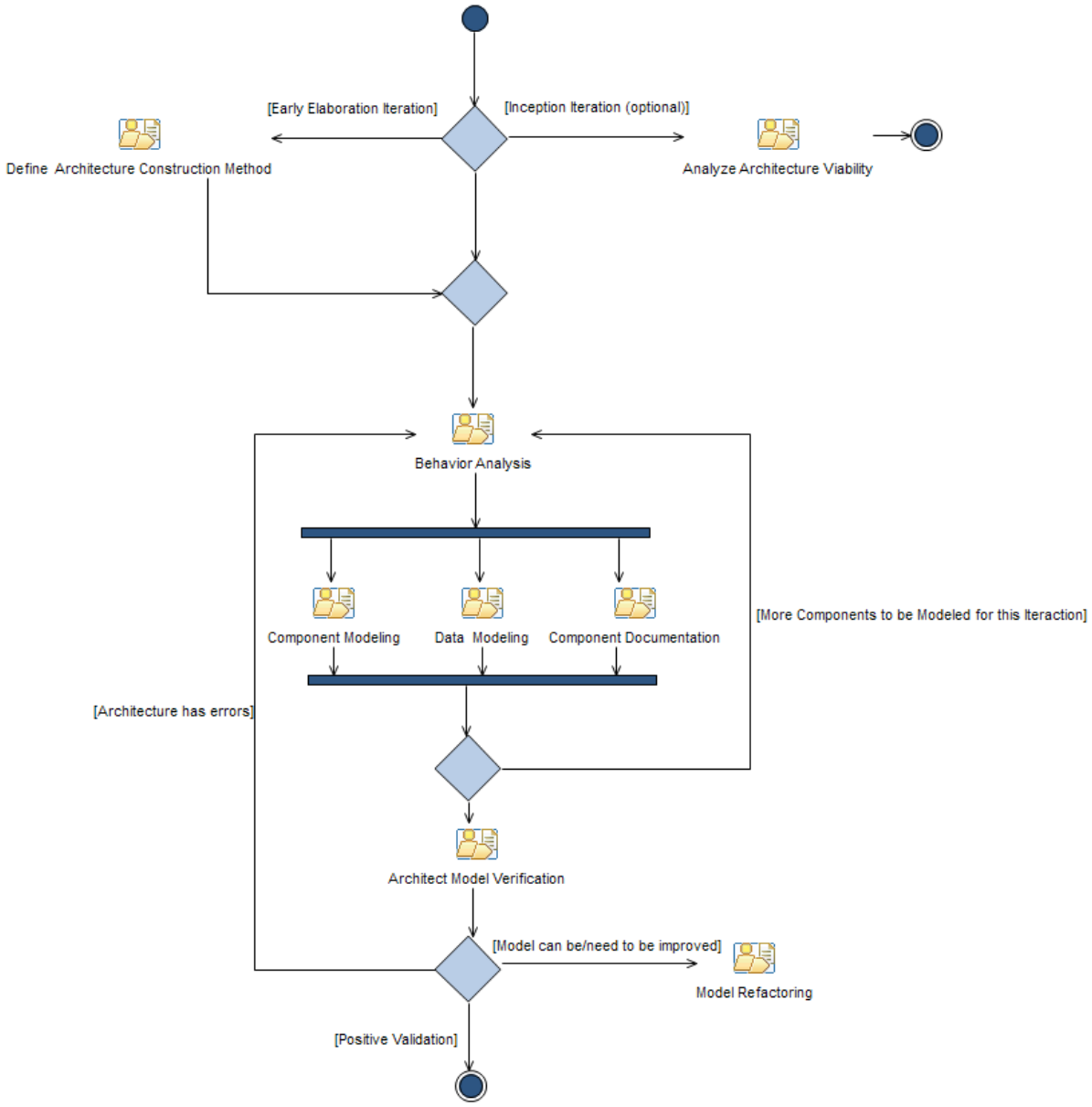

Figura 6.2 - Diagrama de atividades de Análise e *Design*.

A Análise de Viabilidade da Arquitetura (*Analyze Architecture Viability*) é a atividade que se dá na fase de concepção. Essa atividade envolve em fazer uma análise dos requisitos e das necessidades do sistema, construindo um conceito da arquitetura da solução a ser implementada. Essa estrutura conceito é então avaliada e a viabilidade de construir a arquitetura de software é decidida.

Essa atividade é realizada em concepção para que consiga se avaliar se é possível dar continuidade ao projeto. Ela tem caráter opcional, pois sua execução pode ser feita somente pelos analistas com outras informações. Em projetos de maior risco e maior complexidade, os

arquitetos de software podem ter a responsabilidade de auxiliar os analistas com as construções preliminares do conceito. O diagrama de atividade detalhado pode ser observado na Figura 6.3.

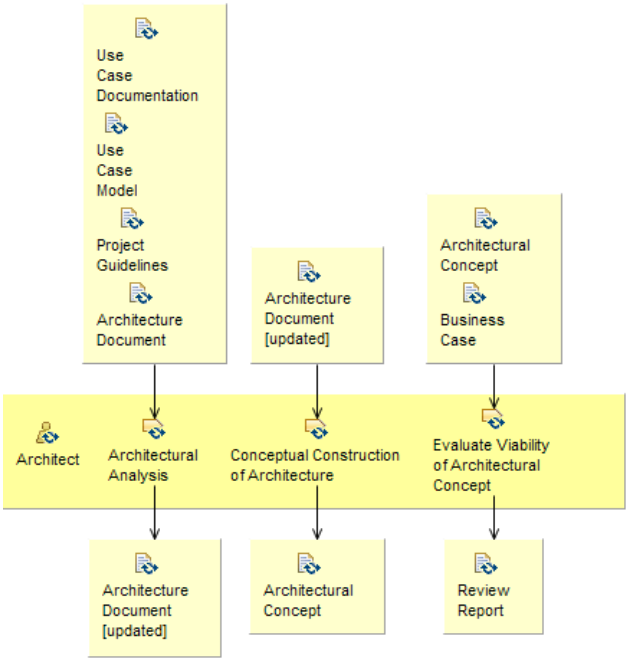

Figura 6.3 - Diagrama de Atividade Detalhado para *Analyze Architectural Viability*.

Nas primeiras iterações da elaboração, é trabalho da equipe de arquitetos definirem qual o método de construção da arquitetura. Na atividade, eles devem analisar os requisitos e descrições do sistema e decidir por uma linguagem de modelagem, especificar a forma como será feita a modelagem e o que será avaliado. Na definição também é trabalho dos arquitetos definirem quais são os propósitos de modelagem do projeto e fazer um planejamento preliminar das iterações que cercam o processo de desenvolvimento da arquitetura. O diagrama de atividade detalhado é apresentado na Figura 6.4.

Uma vez planejados os métodos de construção da arquitetura, passa-se a construção e modelagem do sistema. A primeira atividade é analisar as visões e requisitos do sistema e sintetizar os comportamentos existentes em elementos e componentes computacionais conceituais. A intenção da Análise de Comportamento (*Behavior Analysis*) é reconhecer os comportamentos de software e gerar um guia conceitual do que os modelos de software

precisam. É a partir da Análise de Comportamento também que o esboço da interface de usuário é realizado.

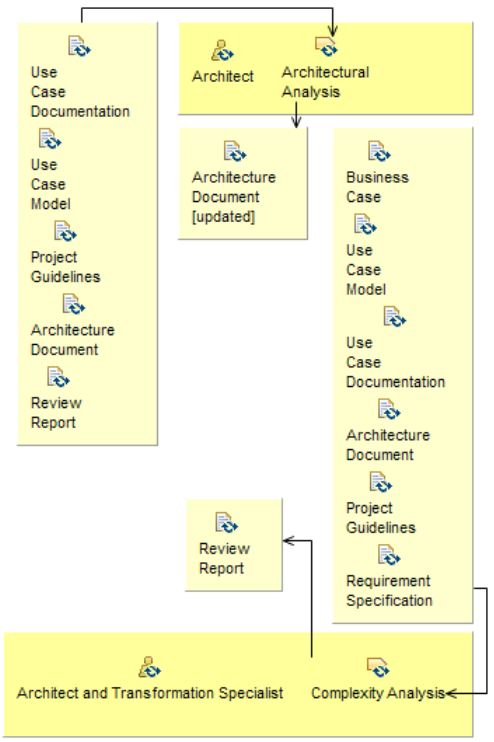

Figura 6.4 - Diagrama de Atividade Detalhado para *Define Architecture Construction Method*.

O principal objetivo dessa atividade é garantir que exista uma compreensão dos requisitos e dos processos. Também se busca que os arquitetos tenham conhecimento da necessidade de informações e compreendam corretamente as relações existentes no sistema. Sendo assim, o desenvolvimento de modelos entra em conformidade com os objetivos de Mohagheghi et al. (2009) e com os critérios expostos por Lange e Chaudron (2005).

Após ter feito a análise, os arquitetos de software podem então modelar e documentar os componentes para iteração corrente. A modelagem de componentes é onde os arquitetos representam visualmente os aspectos existentes do software. Essa atividade é complementada pelo artefato trabalhado pela documentação, que dá a limitação e detalha as informações pertinentes para que ocorra a transformação M2M ou M2T.

Dadas às necessidades do sistema, os arquitetos podem também precisar modelar os dados do sistema. A atividade de Modelagem de Dados (*Data Modeling*) representa a criação

de bancos de dados e o cuidado com a descrição e especificação dos dados que são utilizados pelo sistema. Em MDE é importante não deixar nada ambíguo em relação aos componentes e dados, já que transformações não conseguem lidar bem com ambiguidade. Assim, a atividade busca atingir objetivos de completude e corretude.

O ciclo de desenvolvimento de modelos continua durante a iteração até que todos componentes planejados tenham sido modelados e especificações. No desenvolvimento orientado a modelos, a arquitetura deve estar com um certo grau de robustez antes de passar para próximas iterações, sem um modelo com um nível correto de informações, a transformação parcial do sistema não pode ocorrer.

Após a modelagem, os arquitetos devem testar seus próprios modelos para realizar a primeira validação dos modelos. A intenção é reconhecer erros que possam ter ocorrido durante a modelagem e validar que os objetivos de modelagem foram completamente atingidos. Também é importante que o modelo verifique a efetividade dos modelos, buscando pontos onde podem ser melhorados. O diagrama de atividade detalhado para essa atividade é apresentado na Figura 6.5.

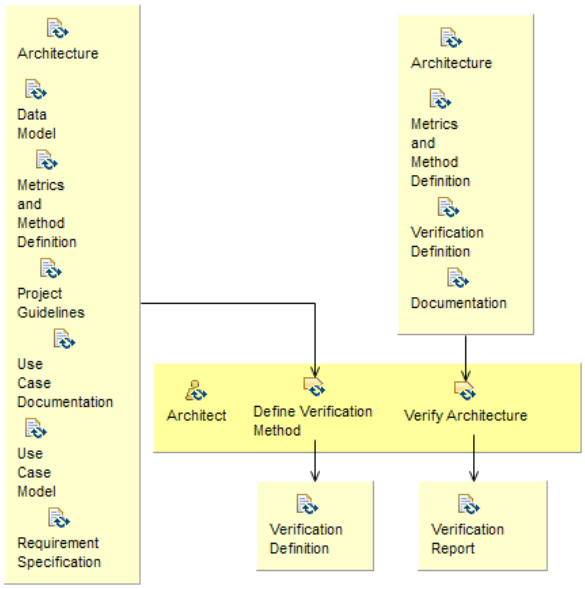

Figura 6.5 - Diagrama de Atividade Detalhado para *Architect Model Verification.*

Caso haja erros, é necessária uma análise de comportamento nova, incluindo os modelos construídos como artefato de entrada para buscar quais erros foram cometidos e onde os modelos devem ser modificados. Se o modelo pode ou precisa ser melhorado, não estando errado, os arquitetos podem decidir refatorar o modelo. A refatoração é importante, pois consegue sintetizar e dar maior efetividade aos modelos. Pode-se ressaltar que tempo é uma questão que os arquitetos devem analisar antes de refatorar. O diagrama de atividade detalhado é apresentado na Figura 6.6.

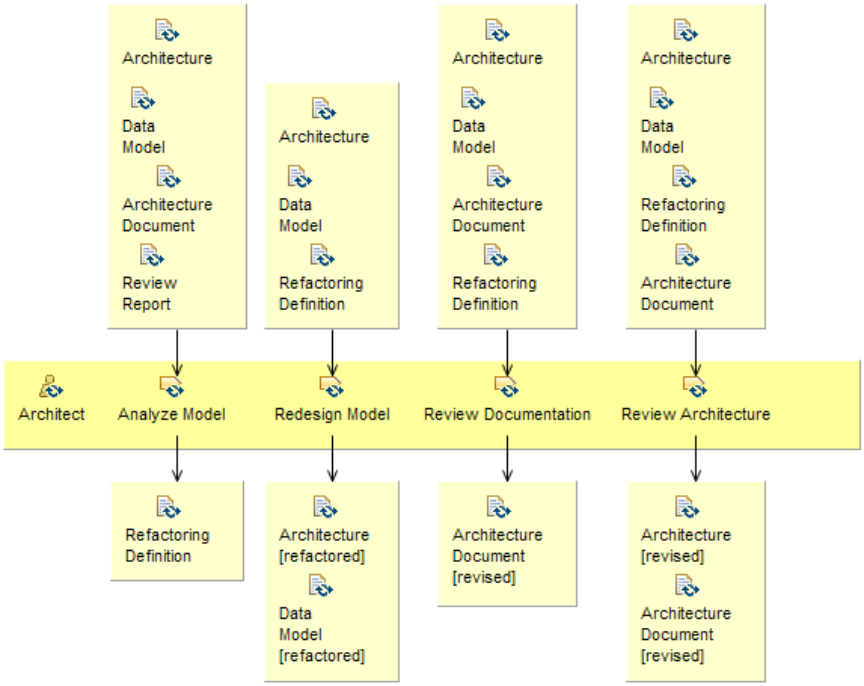

Figura 6.6 - Diagrama de Atividade Detalhado para *Model Refactoring*.

Uma vez que os arquitetos decidirem que os modelos não têm erros ou melhorias a serem feitos durante a iteração atual, e que a arquitetura de sistema atual está estável, eles podem terminar a iteração de modelagem. Ao fim do processo, a disciplina resulta em uma arquitetura de sistema parcial ou completa e validada preliminarmente. A validação feita pelos arquitetos não é a final, no entanto. Assim como em desenvolvimento tradicional, não é ideal que os arquitetos sejam os únicos a avaliar os próprios modelos, já que eles podem ser tendenciosos na avaliação. Essa questão é discutida na disciplina de teste.

6.3 Gerenciamento de Transformações

Essa disciplina é a que substitui a implementação. Por não existir a criação de um código para gerar o sistema, as partes do processo que envolvem programação de software e aplicação da arquitetura deixam de ter sentido.

O processo de programação é automático, realizado pela transformação. Como já discutido, o processo de MDE não abole a prática de codificação. Sua prática, porém, é diferenciada de um processo tradicional de software.

A disciplina de gerenciamento de transformações (*Transformation Management*) é introduzida, então, para substituir a implementação. É ela quem vai incorporar todos os aspectos que revolvem a transformação de modelos para código. A disciplina ocorre em três fases do desenvolvimento, somente sendo deixada de lado na fase de concepção.

É importante ressaltar que as atividades realizadas dentro da disciplina não são atreladas somente a um projeto de software, tendo uma execução mais global, envolvendo todos os projetos correntes, do que uma especializada para cada projeto. Na figura 6.7 pode ser observado o diagrama de atividades para gerência de transformações.

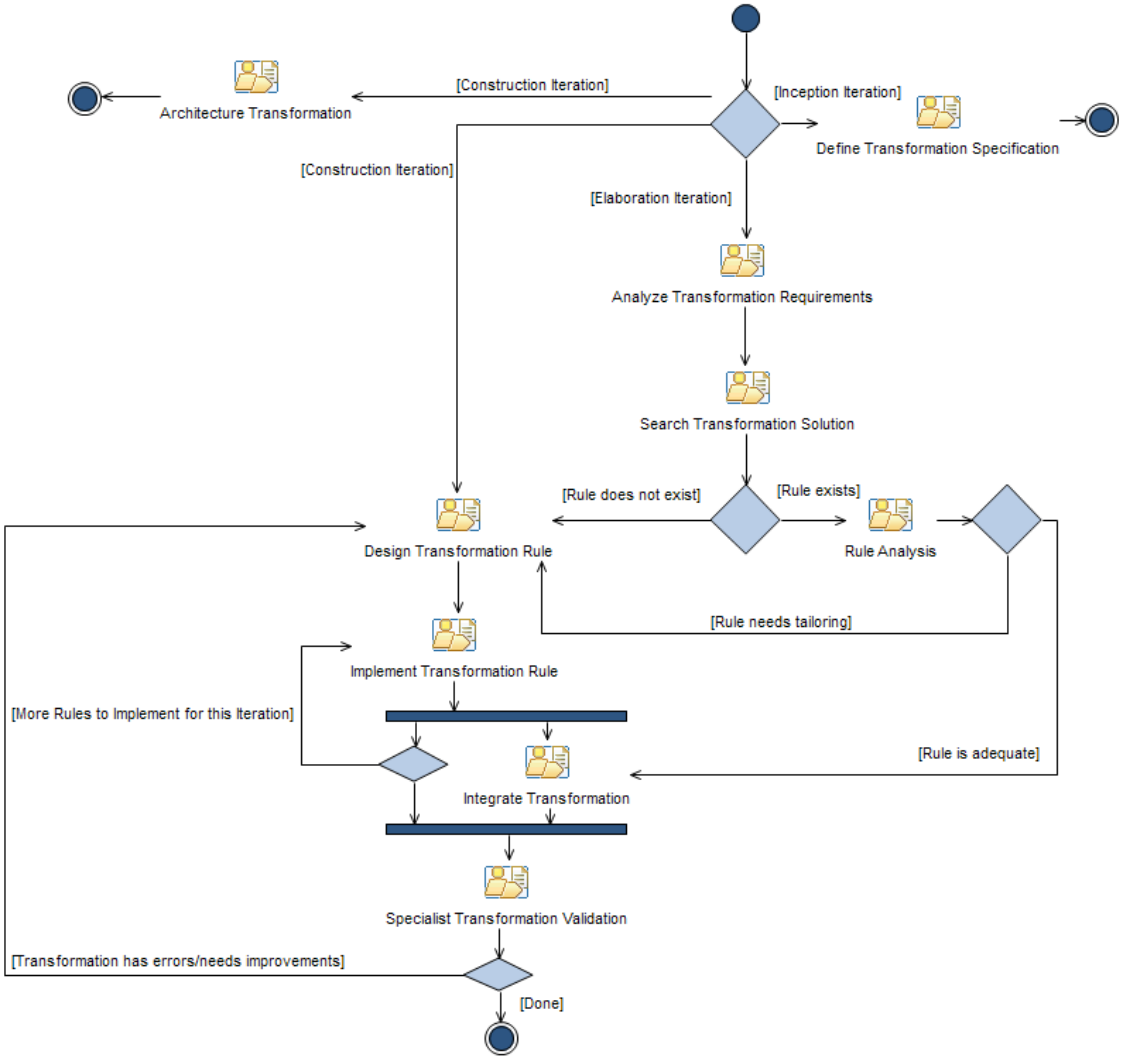

Figura 6.7 - Diagrama de atividades para Gerenciamento de Transformações.

Na fase de concepção, definem-se quais ferramentas serão utilizadas para transformação de modelos e qual linguagem para construção de regras de transformação será utilizada. Essa decisão é feita a partir do reconhecimento das necessidades que os projetos desenvolvidos têm e quais as habilidades da equipe com as ferramentas. No caso do especialista de transformações, a decisão influencia o ambiente em que ele trabalhará, então é importante que a tecnologia utilizada seja de conhecimento dele. O diagrama de atividade detalhado é apresentado na Figura 6.8.

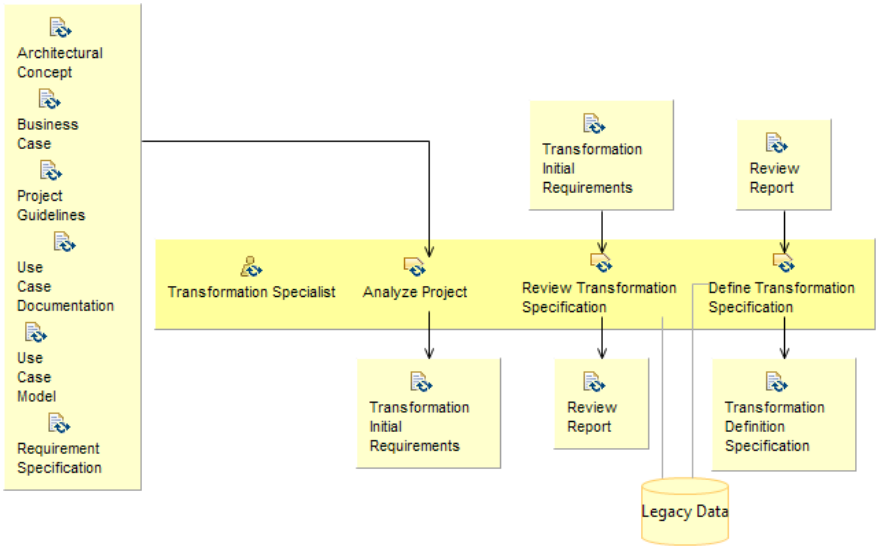

Figura 6.8 - Diagrama de Atividade Detalhado para *Define Transformation Specification*.

Escolher a especificação correta é importante, pois ela é quem dará forma e guiará o desenvolvimento de software e da arquitetura. Sendo assim, a decisão é uma atividade que é realizada em conjunto pelos analistas, arquitetos e especialistas em transformações.

Na fase de construção, durante o ciclo de desenvolvimento do sistema, a disciplina trata da transformação da arquitetura em código. Os arquitetos e o especialista em transformações submetem a arquitetura do sistema, seja ela parcial ou completa, ao processo de transformação. O resultado é uma *build* do sistema ou uma versão finalizada.

As outras partes do ciclo de atividades apresentados na figura remetem-se ao processo de desenvolvimento e adaptação das transformações. Normalmente uma equipe não irá utilizar tecnologias diferentes durante o desenvolvimento de vários projetos. Existe também o fato de que o desenvolvimento de transformações não é atrelado a um projeto ou um ciclo de desenvolvimento.

Haja vista que os componentes das transformações são construídos com o intuito de serem reutilizados, percebe-se a necessidade de ter um ciclo independente de desenvolvimento de transformações. Sendo assim, as atividades representadas após a Análise de Requisitos de Transformação (*Analyze Transformation Requirements*) fazem parte de um ciclo que faz parte do processo de MDE, porém é paralelo a esse. O diagrama de atividade detalhado pode ser observado na Figura 6.9.

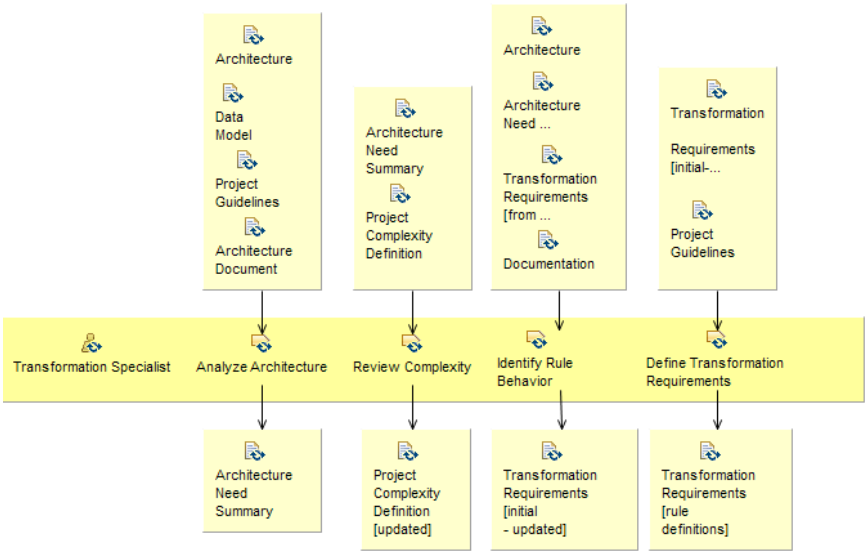

Figura 6.9 - Diagrama de Atividades Detalhado para *Analyze Transformation Requirements*.

A primeira atividade tem, portanto, o objetivo de reconhecer os requisitos e necessidades que os arquitetos e os modelos de sistemas geram para a transformação. Esses requisitos são retirados e analisados a partir do desenvolvimento da disciplina de Requisitos e Análise e Design, isto é, dos modelos e das definições sobre a arquitetura do sistema.

Uma vez que o documento de requisitos de transformação é elaborado e o especialista tem conhecimento das necessidades da arquitetura, ele deve realizar uma pesquisa por regras (ou soluções) de transformações que satisfaçam os requisitos. O especialista pode buscar essas regras em bases conhecidas da tecnologia utilizada ou a partir do sistema legado de transformação. O diagrama de atividade detalhado é apresentado na Figura 6.10.

Se a regra existe, o especialista em transformações deve analisar a regra de transformação, seu código e funcionamento. A partir do parecer, o especialista consegue reconhecer se é necessário adaptar a regra para o sistema construído ou se ela é adequada.

Para regras não existentes inicia-se então um processo de desenvolvimento de regras de transformação. Esse processo também serve para regras que necessitam passar por processo de adaptação e aquelas que precisam ser revisadas.

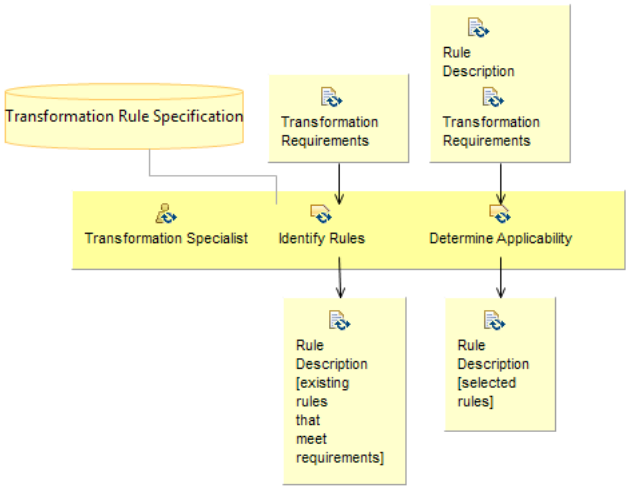

Figura 6.10 - Diagrama de Atividade Detalhado para *Search Transformation Solution.*

A primeira atividade é projetar a regra de transformação. O especialista deve compreender o processo de transformação e modelar como será a implementação de sua regra. Ele também deve procurar documentar os componentes a serem construídos ou modificados. Ao fim da atividade ele deve ter um projeto de regra de transformação e um guia para a implementação. O diagrama de atividades detalhado pode ser visto na Figura 6.11.

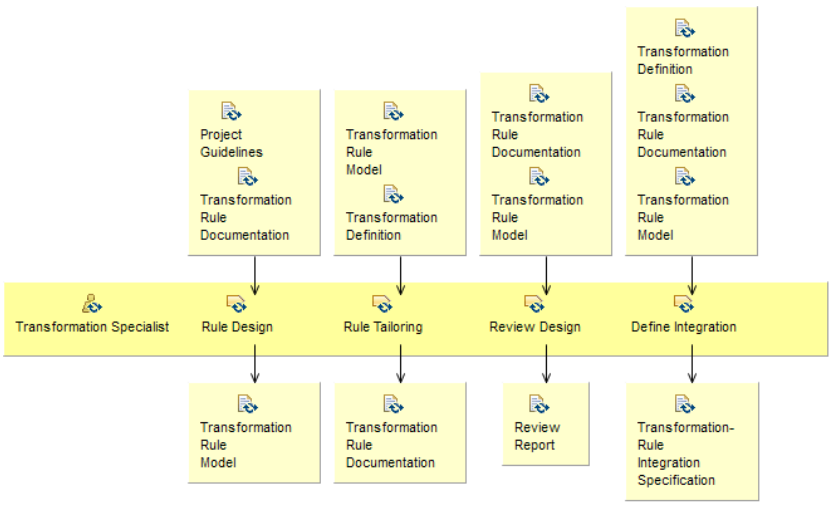

Figura 6.11 - Diagrama de Atividades Detalhado para *Design Transformation Rule*.

Na atividade de Implementação de Regra de Transformação (*Implement Transformation Rule*) o especialista em transformações programa a regra planejada utilizando

a linguagem de transformação especificada nas fases de concepção. O especialista aqui se torna um programador. Ele deve criar rotinas e formas para que as ferramentas interpretem a arquitetura prevista.

O especialista em transformações enquanto programador ainda deve Integrar a Transformação (*Integrate Transformation*). É responsabilidade deste papel pegar os elementos e a transformação existente e integrar com as novas regras de forma consistente e estável, seja ela adaptada, construída desde o início ou somente reutilizada de uma solução existente.

Após a integração, o especialista deve fazer então sua validação da transformação. Esse processo ocorre de forma similar à validação de modelos apresentada em Análise e Design. O especialista avalia as regras que ele mesmo implementou para identificar erros ou melhorias que podem ser feitas, caso seja feito, o processo retorna ao ciclo de projetar a regra de transformação. O último passo do processo passa por teste.

É importante ressaltar que se procurou inserir conceitos encontrados em Vanhooff et al. (2007) nessa disciplina. São esses os autores que melhor definiram os processos genéricos que ocorrem em transformações no desenvolvimento MDE, sendo assim, o fluxo e a as atividades se baseiam em conceitos colocados por eles.

6.4 Teste

A disciplina de teste procura avaliar e validar o sistema construído durante o projeto. A partir dos relatórios construídos sobre o sistema, os desenvolvedores podem corrigir erros, ajustar abordagens, consertar funcionalidades, entre outros. O caso de MDE tem algumas mudanças particulares.

As partes de teste de *build* e versões de usuário do software continuam ocorrendo normalmente. O processo de MDE, porém, tem uma automatização muito grande e o teste passa a ser inserido em vários momentos durante o desenvolvimento, não envolvendo somente a avaliação do produto parcial ou pronto, mas da arquitetura e da transformação que cria a *build* a ser avaliada.

Somente criar modelos, conforme visto anteriormente, não significa que os mesmos foram construídos com qualidade. Como o desenvolvimento é muito dependente da arquitetura em MDE e qualquer erro ou problema na modelagem tem grande impacto na transformação, os modelos acabam sendo foco de teste também.

A transformação, conforme visto na seção anterior, passa por um processo de desenvolvimento que se assimila ao processo tradicional de desenvolvimento de software. Para que se possa assegurar que as regras adaptadas, geradas ou integradas à transformação sejam precisas e confiáveis, elas também devem passar por teste.

Sendo assim, a disciplina de teste passa a ter três dimensões. As dimensões de teste de transformação e do sistema seguem um fluxo similar. A dimensão de modelos se caracteriza por ser um pouco diferente. Na figura 6.12 é mostrado o diagrama de atividades para teste.

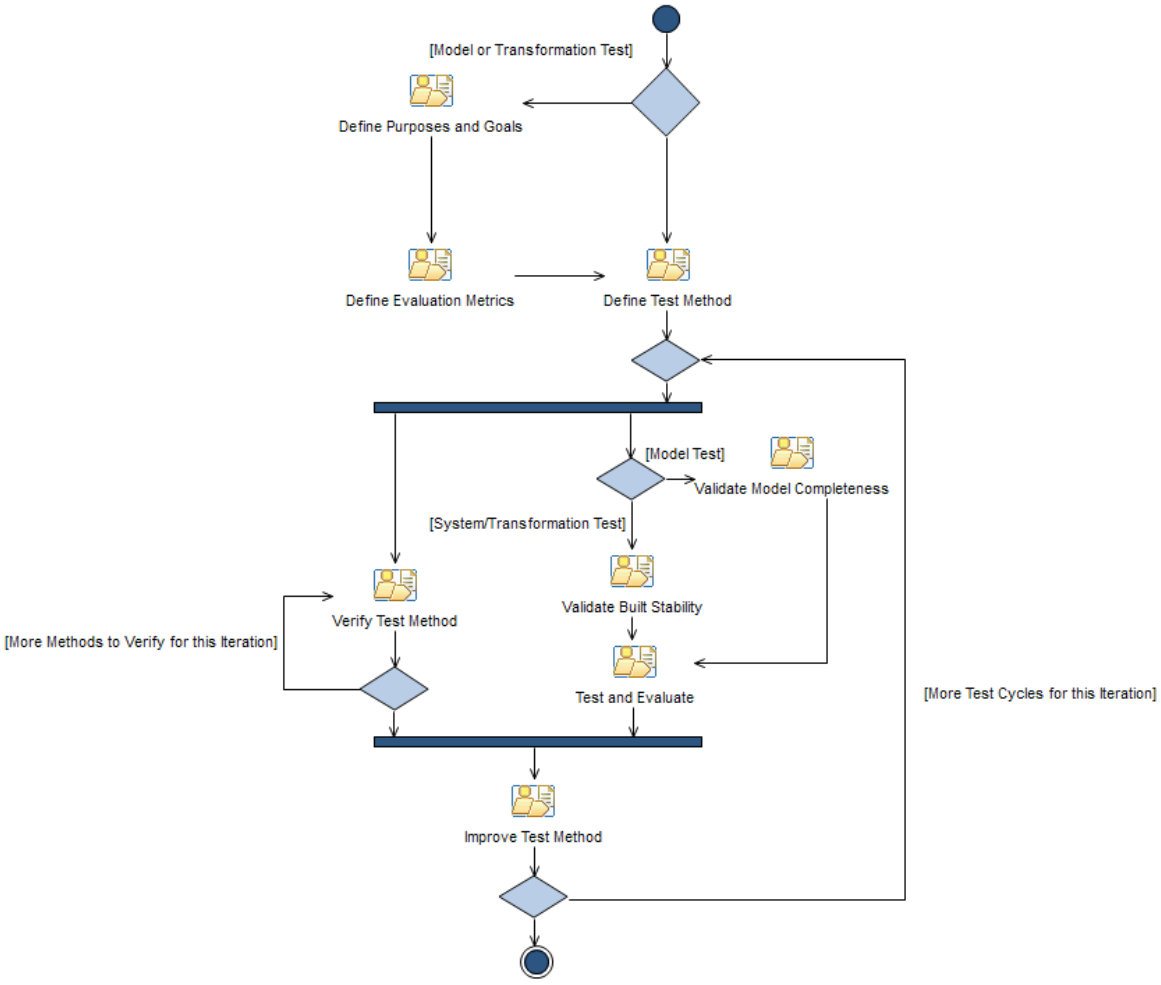

Figura 6.12 - Diagrama de Atividades para disciplina de Teste.

As atividades de teste são realizadas durante as várias fases do ciclo de desenvolvimento. Elas avaliam os artefatos resultantes de outras disciplinas e devolvem um *feedback* sobre as estruturas construídas. Por ter três dimensões diferentes para teste, a disciplina em MDE passa a ter alguns ajustes que devem ser considerados para cada uma delas.

Se o artefato a ser avaliado é a arquitetura (envolvendo modelos) ou a transformação, os testadores devem primeiro determinar os propósitos e objetivos que esses elementos têm e quais são os objetivos ao testar esses elementos. No caso de modelos, os testadores também devem definir o nível de abstração da arquitetura alvo de teste.

Modelos mais abstratos podem ser submetidos a métodos qualitativos e podem ser validados de forma mais fácil (e menos precisa). A abstração não permite que se tenham dados quantitativos, além disso, a validação qualitativa é mais fácil de alcançar. Já modelos pouco abstratos permitem uma avaliação mais qualitativa e precisa (devido ao volume de dados quatitativos) sobre os elementos da arquitetura, porém sua validação é mais difícil. O diagrama de atividade detalhado para essa atividade é demonstrado na Figura 6.13.

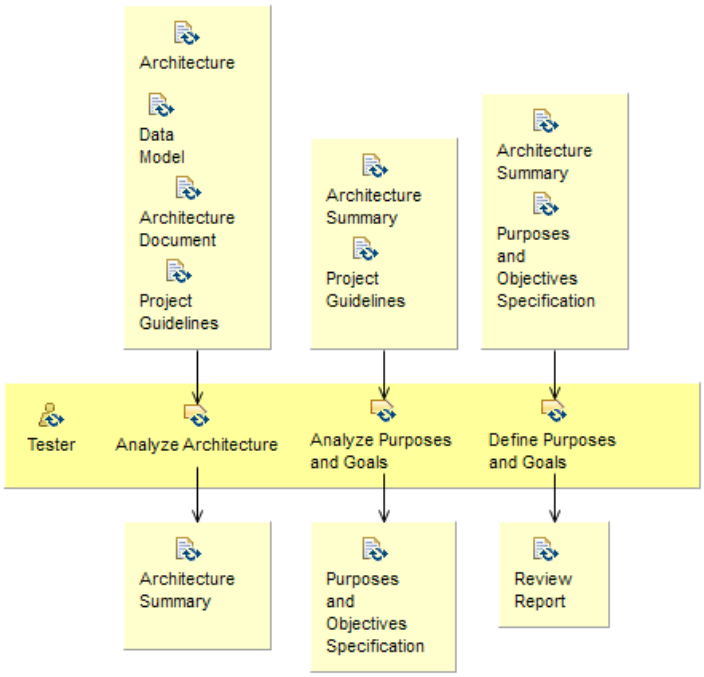

Figura 6.13 - Diagrama de Atividades Detalhado para *Define Purposes and Goals*.

Uma vez definido os propósitos e objetivos, os testadores então escolhem as métricas a serem utilizadas para avaliar modelos ou a transformação. A escolha das métricas corretas para modelos é crucial ao processo de teste dos mesmos. A relação entre métrica e modelo ainda é abstrata em muitos casos, conforme já discutido. Logo, a escolha de métricas com pouco entendimento pode levar a um teste que não dê um relatório claro sobre os modelos.

No caso de transformações, a definição de métricas devem levar em conta as características relacionadas às transformações, incluindo as estruturas de modelos que serão transformadas por ela. Assim, a utilização de um método como GQM, como definido por Rahimi e Lemo (2011), é interessante, já que este engloba essas questões sobre transformação. O diagrama de atividade detalhado pode ser observado na Figura 6.14.

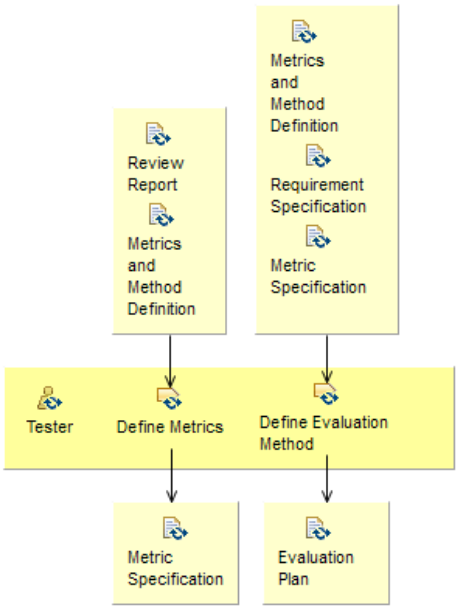

Figura 6.14 - Diagrama de Atividade Detalhado para *Define Method and Metrics*.

A atividade de Definir Método de Teste (*Define Test Method*) ocorre para as três dimensões. No caso de ser um teste de modelo ou de transformações, as atividades precedentes servem como guias para o método de teste que será utilizado. No caso de avaliar *builds* do sistema, os testadores devem identificar qual será o foco do teste para a iteração atual e então definir um método de teste. Uma vez definido os métodos, os testadores devem documentar e elaborar as estratégias de teste.

Caso o teste seja de modelos, não há sentido em validar a estabilidade da *build*. Essa atividade tem objetivo de verificar se o alvo de teste é estável suficiente para ser submetido a testes, impedindo, assim, o desperdício de recursos do projeto em testes sem resultados. Enquanto essa atividade se mantém para o caso de analisar uma *build* do sistema ou o software de transformação, isso não é verdade para modelos.

Sendo assim, a atividade que substitui a validação da estabilidade da *build* em modelos é a Validar Completude de Modelo (*Validate Model Completeness*). Essa atividade tem o mesmo objetivo daquela. A intenção é que não se desperdice recursos do projeto avaliando arquitetura e modelos que não sejam o suficientemente robustos para gerar resultados relevantes. O diagrama de atividade detalhado é mostrado na Figura 6.15.

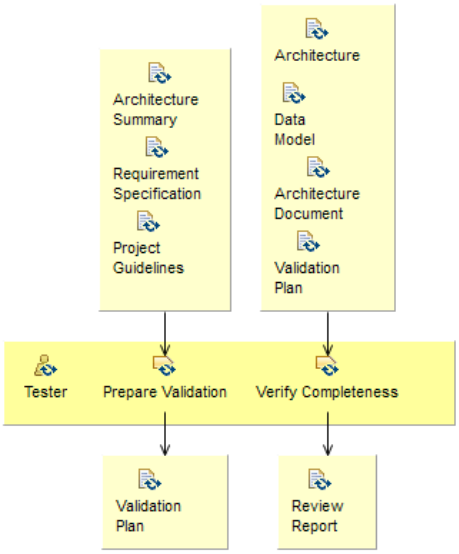

Figura 6.15 - Diagrama de Atividade Detalhado para *Validate Model Completeness*.

Uma vez que os testadores se certificam de que os elementos são testáveis, eles podem executar o teste conforme o método e as métricas definidas anteriormente. Dependendo da tecnologia usada para a modelagem (linguagem de modelagem) os testadores podem realizar testes executáveis em modelos, algo que a xUML permite, por exemplo.

As outras atividades envolvidas em teste se mantêm. Deve-se verificar que o método de teste escolhido é apropriado para o uso que os testadores pretendem dar e que os resultados gerados serão relevantes.

Outro aspecto da disciplina de teste é o melhoramento dos aspectos envolventes do método de teste. A partir da adaptação do método e sua execução durante o teste, os testadores criam uma base de dados que pode servir como melhoramento e guias para testes futuros, melhorando o método.

6.5 Ambiente

A disciplina de ambiente representa a customização do processo de desenvolvimento de software e das ferramentas para o projeto. Em MDE, o ambiente envolve a estruturação do processo que integra a transformação com o ciclo de desenvolvimento de software.

Como já dito, o desenvolvimento de transformações é universal para todos os projetos e equipes que estejam trabalhando. Sendo assim, ele não se foca somente em um projeto, fazendo parte de todos.

É importante que a forma como o desenvolvimento de transformações se integra ao ciclo de software seja especificado. Para isso, o gerente de ambiente deve analisar o projeto, a equipe de trabalho e realizar as adaptações. O diagrama de atividades é apresentado na figura 6.16.

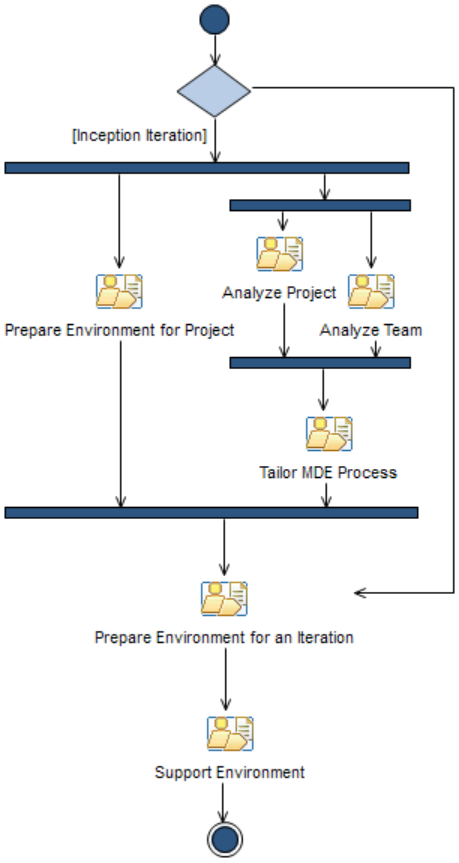

Figura 6.16 - Diagrama de Atividades para Ambiente.

Por fim, o gerente deve utilizar os resultados de sua análise para estruturar o processo de MDE para o projeto atual. O principal foco da adaptação é integrar o ciclo de desenvolvimento do sistema com o ciclo de desenvolvimento de transformações. Sendo assim, o artefato resultante da adaptação do processo de MDE serve como direcionador para o processo do projeto e para o ciclo do desenvolvimento de transformações. O diagrama de atividade detalhado para *Tailor MDE Process* é demonstrado na Figura 6.17.

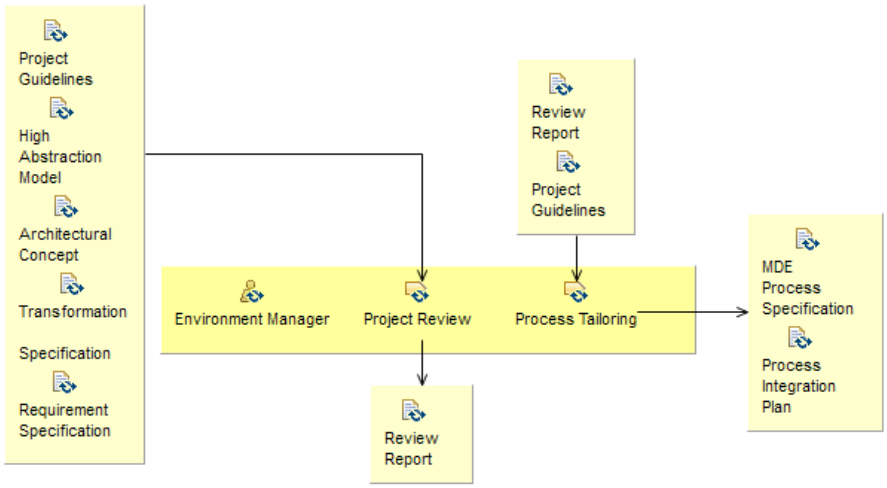

Figura 6.17 - Diagrama de Atividade Detalhado para *Tailor MDE Process*.

A integração de transformação com o projeto envolve em como será feita a entrega de requisitos pela equipe e como os especialistas auxiliarão com o desenvolvimento. Também é parte da integração a adaptação das ferramentas de transformação às necessidades do projeto.

6.6 Papéis

Dentro de um processo, os papéis são descrições que encapsulam comportamentos específicos dentro de um processo. Os papéis são utilizados por atores. Ator é um indivíduo da equipe de desenvolvimento e participante do processo de desenvolvimento de software. Os atores podem assumir mais de um papel durante o processo de desenvolvimento, porém não podem atuar em dois papéis distintos ao mesmo tempo.

Os papéis são descritos através de quais responsabilidades eles têm e quais atividades que eles desempenham. Também faz parte da descrição dos papéis um grupo de habilidades que o ator que desempenha o papel deve ter para desempenhá-lo.

Utilizando os conceitos de MDE percebe-se que existem papéis que perdem sua significância, outros que mudam suas responsabilidades e novos papéis são criados. Também se podem citar papéis que não mudam suas responsabilidades ou sua descrição.

Os papéis que não mudam sua descrição, responsabilidades e habilidades necessárias, são aqueles que trabalham com atividades que não são afetadas pelos conceitos dos novos paradigmas. Um exemplo deste tipo de papel é o analista. O analista usualmente é a pessoa que representa as preocupações do consumidor e dos usuários finais, utilizando-se de informações que obtém dos *stakeholders* e dos clientes. É ele quem deve criar a visão do sistema e documentar os requisitos do projeto para a equipe. Ele também auxilia na construção de modelos mais abstratos, como o modelo de casos de uso.

Nele se exemplifica a estabilidade de alguns papéis, por trabalhar com os requisitos e os clientes, suas atividades e os artefatos que ele trabalha são os iniciais e que guiam o processo de desenvolvimento. As tarefas que ele deve desempenhar, então, podem ser colocadas como não influenciadas pela mudança que o paradigma propõe.

Abaixo são apresentados os papéis básicos do processo. Ressalta-se que a definição dos papéis na tabela é abrangente, sendo que um papel pode ser especificado em papéis menores, se necessário. Os papéis mais importantes do processo de desenvolvimento em MDE são:

- Analista;
- Arquiteto;
- Especialista em Transformação;
- Testador;
- Gerente de projeto; e
- Gerente de ambiente.

Nas próximas subseções, as características de cada um dos papéis são apresentadas. É importante colocar que existem papéis que não são discutidos por duas razões: (i) sua descrição não muda com o novo paradigma; e (ii) são papéis que mudam conforme o tipo e complexidade do sistema.

Assim, procurou-se agregar as responsabilidades e atribuições em papéis mais genéricos. Na próxima subseção são explicadas quais especificações podem existir dentro de um papel e seus comportamentos.

Alguns papéis, apesar de ter pouca mudança, são relembrados aqui por interferirem no processo direto de desenvolvimento de software em MDE. Pode-se usar como exemplo

novamente o papel de analista, que pouco mudam suas atividades, porém ele é de crucial importância para o ciclo de desenvolvimento de software.

6.6.1 Analista

O analista, conforme já discutido, é um papel que se mantém quase imutável no processo de MDE. Suas responsabilidades ainda se encontram em representar as preocupações dos clientes e consumidores finais.

Cabe ao analista compreender o que desejam os *stakeholders*, qual é o domínio do negócio e os requisitos que representam as necessidades percebidas. Ele então tem responsabilidade de documentar isso de forma inteligível para a equipe de desenvolvimento. Sendo assim, o analista pode ser subdividido em duas especificações diferentes, são elas:

- Analista de Sistema; e
- Analista de Negócio.

O analista de negócio representa as responsabilidades do analista em termos de identificar os problemas e as oportunidades dentro do domínio de negócio. Ele também deve conhecer o negócio e a tecnologia utilizada, ou ter capacidade de absorver tais conhecimentos rapidamente.

Também fica a cargo do analista de negócios modelar o processo de negócio para melhor compreensão das atividades e do fluxo de informações que existem. Ele deve se comunicar com os membros externos e internos ao projeto para adquirir informações sobre o domínio de negócio.

A partir de uma análise do processo de negócio, o analista de sistema realiza o seu trabalho. Junto aos análistas de negócio, sua responsabilidade é articular as necessidades que os *stakeholders* têm dentro do sistema em requisitos de software. Ele também deve utilizar as necessidades do negócio para elicitar os requisitos existentes.

O analista é quem inicialmente classifica os requisitos em termos de funcionais e não-funcionais. Além disso, ele tem a responsabilidade de colocar quais os casos de uso que são prioritários.

O papel do analista como um todo elabora e trabalha com artefatos de desenvolvimento como o glossário, utilizado para padronizar e unificar a linguagem dentro do processo; e o documento de visão, que estabelece quais são as visões dos *stakeholders* em termos da solução a ser definida.

O papel não apresenta mudanças de sua versão do desenvolvimento tradicional em suas responsabilidades centrais. Pode-se destacar, porém, que o analista é colaborador no desenvolvimento em iterações mais a frente do projeto, colaborando com a construção de modelos, isto é, o artefato principal de desenvolvimento, e supervisionando a implementação dos requisitos de maneira correta.

Como habilidades requeridas ao analista estão: facilidade na identificação e entendimento de problemas e oportunidades, conseguir articular e identificar necessidades com os problemas centrais a serem resolvidos, reconhecer oportunidades dentro do modelo de negócio, comunicar-se bem com membros internos e externos ao projeto e ter conhecimento tecnológico e de negócio.

6.6.2 Arquiteto

O arquiteto é o papel central a abordagem MDE. O arquiteto absorve o papel do desenvolvedor e se torna aquele que realiza as atividades principais de desenvolvimento do sistema.

O arquiteto é o indivíduo que deve coordenar o projeto técnico do sistema e realizar as decisões importantes sobre a arquitetura de software, a qual é artefato central do desenvolvimento de software no paradigma MDE. Isso faz com que esse papel seja responsável pelos artefatos principal de desenvolvimento.

O arquiteto deve trabalhar junto com os analistas para assegurar que a arquitetura do sistema está em conformidade com os requisitos e necessidades do negócio. Entre suas responsabilidades, podem-se colocar as seguintes especificações ao papel:

- *Designer*;
- Especialista em qualidade de modelos;
- Avaliador de modelos;
- Documentalista; e
- Revisor de modelos.

O papel de arquiteto cresce consideravelmente em MDE, podendo ser dividido em duas dimensões. Em uma dimensão de projetista, o arquiteto é responsável por idealizar uma arquitetura abstrata do sistema, como ocorre no desenvolvimento comum. Em outra dimensão de desenvolvedor, ele é responsável por especificar essa arquitetura e documentá-la com as informações pertinentes para garantir que a transformação ocorra.

Haja vista essas características, o arquiteto deve estar sempre se comunicando com os especialistas em transformação. Uma habilidade essencial ao arquiteto é conhecer as ferramentas de transformação escolhidas e os poderes das mesmas (o que conseguem e não conseguem interpretar).

Sendo assim, se classificam os comportamentos mais específicos do arquiteto. Como *designer*, o arquiteto deve modelar o sistema para representar a arquitetura do sistema. Esses modelos incluem metamodelos abstratos e modelos de desenvolvimento (menos abstratos).

O comportamento de especialista em qualidade em modelos tem dois objetivos. Primeiramente é descrever que o arquiteto tem que ter conhecimento das práticas e técnicas para assegurar a construção de bons modelos, conforme discutido na subseção 3.2.2. Outro objetivo é que ele consiga realizar validação primária dos modelos e realizar a refatoração de modelos.

Como documentalista, é objetivo do papel documentar a arquitetura do sistema com informações pertinentes ao processo. O comportamento de revisor de modelos é usado durante a elaboração da arquitetura, onde um trabalho em conjunto dos arquitetos leva a uma arquitetura revisada e, provavelmente, mais estável.

6.6.3 Especialista em transformações

O especialista em transformações tem papel central de suporte para MDE. É nesse papel que se reúnem as responsabilidades em gerar, gerenciar e adaptar as regras de transformação para um melhor processo de transformação. Algo que se pode ressaltar sobre esse papel é que ele tem em suas especificações a responsabilidade de trabalhar com código, assim como um programador em um desenvolvimento comum.

A definição do especialista em transformações é um indivíduo que trabalha em um ciclo de desenvolvimento integrado das regras de transformação. Suas funções permitem o andamento que o sistema projetado seja transformado. Ele trabalha junto aos arquitetos e analistas de projetos.

Sua especificação de subpapéis foi baseada na definição de Vanhooff et al.(2007), a qual descreve o processo de desenvolvimento de transformações. Sua especificação é:

- Analista de transformações;
- Arquiteto de transformações;
- Desenvolvedor de transformações; e
- Testador de transformações.

O especialista em transformações tem como responsabilidade receber e analisar as informações e requisitos que recebe dos analistas e arquitetos sobre o projeto e o sistema a ser construído. A partir da análise ele pode identificar quais regras e adaptações serão necessárias para o projeto.

Dentro de suas responsabilidades também está planejar e projetar as regras de transformação (no caso de adaptação ou desenvolvimento) e integração das mesmas à tecnologia utilizada pelo projeto. Ele também é o indivíduo que implementa as regras projetadas, representando comportamento de desenvolvedor de transformações. Outra responsabilidade que pode ser especificada é a de realizar testes e validações primárias às regras de transformação desenvolvidas, adaptadas ou integradas.

O especialista em transformações deve ter habilidade com implementação de regras de transformação, conhecimento do processo de MDE, de como as regras funcionam e de

construção de arquitetura conforme os métodos utilizados para construção do sistema. Outra habilidade que o especialista deve ter é a de comunicação e abstração, para realizar as atividades relacionadas à análise de transformação e que envolvam outros membros da equipe.

6.6.4 Testador

Esse papel encapsula as necessidades de medir e avaliar o software. Usualmente entre as atribulações e objetivos do papel se incluem (PRESSMAN, 2006; SHUJA e KREBS, 2008):

- Identificar os testes que são necessários;
- Identificar a método mais apropriado para se realizar um teste;
- Planejar os testes;
- Executar os testes;
- Documentar os resultados;
- Analisar os resultados obtidos; e
- Comunicar os resultados dos testes para a equipe.

Isso ainda é verdadeiro em MDE. Existem aspectos que mudam a forma em que os testadores são definidos. Os testadores também devem testar o produto ou parte dele após a transformação. A diferença vem que os testadores devem testar os modelos e a transformação em MDE. Assim se tem duas especificações para esse papel:

- Testador de modelo; e
- Testador de software.

A transformação é vista como software. Por esta razão, não se cria uma especificação para teste de transformações.

Abordagens e modelos como xUML (Mellor e Balcer, 2002) foram criados podendo ser testados executando os modelos e torna-los mais testáveis. Os testadores de modelo, no

entanto, podem testar os modelos por outros métodos. Um desses métodos é o *framework* de Lange e Chaudron (2005), por exemplo.

Assim como testadores de software, os testadores de modelo devem planejar e executar, além de emitir um relatório sobre os resultados que encontrou. Como discutido anteriormente, os métodos de validação de modelos são mais qualitativos que quantitativos. Portanto, é um processo que exige mais esforço do testador e exige que ele tenha conhecimento sobre modelagem e métricas de modelos.

O testador de software continua tendo as mesmas atribuições que teria em um ciclo de desenvolvimento tradicional. As habilidades essenciais ao testador de software são:

- Conhecimento das abordagens e técnicas de teste de software;
- Capacidade em emitir diagnósticos e resolver problemas;
- Conhecer o sistema que está testando; e
- Conhecimento da arquitetura do sistema.

Ainda pode-se destacar que os testadores necessitam de conhecimentos do que estão testando. Ou seja, os testadores de modelos precisam entender as técnicas e abordagens de modelagem, e os testadores de software devem entender do processo de transformação e como os códigos são gerados para conseguir guiar melhor seus relatórios às raízes dos problemas.

6.6.5 Gerente de projeto

O papel de gerente de projetos ainda é o mesmo em sua definição. O papel encapsula o comportamento da pessoa que guia e dirige a equipe para o sucesso do projeto. Ele é quem avaliar os riscos e a evolução do projeto.

Aqui se ressalta que ao mudar a forma de como o desenvolvimento ocorre, muda a forma como o gerente de projetos pode controlar e realizar estimativas sobre o projeto. Ao passar de um paradigma estabelecido para o novo paradigma, perdem-se vários artefatos e meios que eram utilizados para esses fins, como código produzido.

Portanto, coloca-se que o gerente deve encontrar novas formas de planejar, dirigir, organizar e controlar o projeto. Outro ponto é como gerir as pessoas que trabalham no projeto, já que funções e atividades mudam, além da estrutura básica da equipe.

Não é objetivo desta dissertação discutir tais temas. É importante, porém, apontar que existem essas mudanças pertinentes ao papel de gerente de projetos no novo ciclo de desenvolvimento de software.

6.6.6 Gerente de ambiente

Nesse papel é descrito o comportamento da pessoa responsável por controlar o ambiente do projeto. Ele tem duas funções: gerenciar as ferramentas e o ambiente de trabalho e adaptar o processo a realidade em que o projeto está inserido.

O gerente de ambiente, junto ao gerente de projeto, deve compreender a complexidade do projeto e as habilidades da equipe de projeto. É crucial que ele compreenda como ocorre o processo de desenvolvimento de transformações.

Como já discutido, o gerente de ambiente ao adaptar o processo, ele modela a forma como a gerência de transformações será integrada ao ciclo de desenvolvimento de software. Portanto, é importante que o gerente de ambiente tenha habilidade e conhecimento de adaptação e integração de processos de software.

6.6.7 Outros papéis e considerações

Existem outros papéis associados ao processo de desenvolvimento de software que não foram definidos durante esta seção. Esses papéis seguem as definições encontradas no *framework* RUP.

Os papéis administradores representam os outros gerentes que podem auxiliar o gerente de projeto a controlar e dirigir o processo de software. Por exemplo, o gerente de entrega é aquele que cuida do processo em relação à entrega do software para os usuários finais e sua implantação.

Ainda podem existir gerentes de custos, gerentes de tempo ou até mesmo de aquisições. Fica a cargo do gerente de projetos e de ambiente a estabelecer as necessidades do projeto ao iniciar o processo de desenvolvimento. Todos os papéis administrativos podem ser concentrados na figura do gerente de projeto também, se assim necessário.

Os papéis relacionados à transição referem-se às atribulações dos atores em termos de transição de desenvolvimento a implantação para usuários finais. Aqui entram uma gama de papéis diferentes como: escritor técnico (para o manual e documentação para usuário); instrutor (para treinamentos); e desenvolvedor de material de instrução (cria material para cursos de treinamentos).

Optou-se por não mudar a definição de papéis como esses encontrados nas especificações do RUP (SHUJA e KREBS, 2008). Os papéis que tem mudanças significativas são centrais ao desenvolvimento, com destaque ao papel de arquiteto, que se torna o desenvolvedor de fato do processo. Também se destaca o desaparecimento do papel de programador e a inserção do papel de especialista em transformações.

Os papéis representados nessa seção, como já dito, podem ser divididos dentro de suas especificações. Um papel que destaca essa versatilidade é o arquiteto. Pelo papel absorver tantas responsabilidades durante o desenvolvimento, ele pode ser fragmentado em várias responsabilidades diferentes.

A utilização ou não das especificações depende do projeto e da equipe. Cabe ao gerente de projetos e sua equipe realizar essas definições na fase de concepção e preparação do projeto.

6.7 Artefatos

Os artefatos são o que é trabalhado no processo. Representa tudo aquilo que é produzido por uma tarefa durante o processo (SHUJA e KREBS, 2008).

Os artefatos trabalhados em MDE têm suas diferenças ao desenvolvimento de software comum. Por exemplo, enquanto a arquitetura é um modelo abstrato e de guia no desenvolvimento comum, em MDE é uma documentação robusta que serve de centro ao desenvolvimento. As mudanças se dão mais na forma em que os mesmos são usados, porém existem os novos artefatos também (regras de transformação e especificação de transformação). Boa parte dos artefatos se mantém, as atividades que os trabalham, porém, mudam.

Existem alguns artefatos que são inseridos no processo. Maior parte refere-se a gerência de transformações. Outros se referem à documentação do processo. Nesta seção são discutidos os artefatos do processo proposto.

6.7.1 Artefatos de Arquitetura

A arquitetura do sistema é a representação da estrutura básica do sistema. Nela é definida a essência do sistema e os elementos centrais do projeto e suas relações. Serve como guia e um *framework* para o desenvolvimento do sistema.

A arquitetura em termos de MDE é composta por três partes essenciais:

- Modelo;
- Documentação; e
- Modelo de dados.

A diferença principal da arquitetura no novo paradigma está no detalhamento da mesma. Normalmente a arquitetura abstrai aquilo que o sistema é e o aproxima de sua forma

conceitual. Enquanto modelos abstratos servem para o paradigma tradicional, em MDE a necessidade de documentar e especificar os modelos é grande.

Assim, a arquitetura enquanto artefato de MDE é considerada como um conjunto de modelos de sistema, modelo de dados e documentação que possam ser transformados, através de uma transformação ou de uma corrente de transformações, em um código executável e testável. Uma arquitetura parcial que não possa ser transformada é vista como um modelo conceitual ou abstrato. Na Tabela 6.4 se demonstram os artefatos relativos à arquitetura do sistema.

Tabela 6.3 - Artefatos relativos a arquitetura do sistema.

Definição	Artefato	Descrição
Abstração do Sistema	Caso de Uso	Encapsula os comportamentos existentes no sistema, identificando funções, fluxo de eventos e relações do sistema.
	Modelo de Casos de Uso	Descreve o comportamento pretendido dentro da arquitetura do sistema. É um modelo de alta abstração
	Modelo Abstrato	Descreve os comportamentos do sistema sem especificar informações. Não é válido para transformação.
Arquitetura	Modelo de Dados	Detalha os dados utilizados pelo sistema e suas relações. Demonstram como os dados são processados. Objetivam transformação M2T
	Modelo Específico	Detalham os comportamentos do sistema inserindo informações pertinentes à transformação. Objetivam transformação M2T.
	Documentação Técnica	Descrevem informações que completam o modelo. Objetivam auxiliar as ferramentas de transformação M2T
	Teste Primário de Modelo	Teste realizado pelos arquitetos para validar seus modelos antes da transformação.

Os modelos abstratos ainda existem para guiar a construção da arquitetura. Um exemplo disso são os modelos de caso de uso, que são úteis para identificação e encapsulamento dos comportamentos de software. Através dos casos de uso também se expressam as necessidades encontradas nos requisitos.

6.7.2 Artefatos de Transformação

A transformação consiste de vários novos artefatos. Por ser um processo de desenvolvimento dentro do processo de desenvolvimento do sistema, ele é construído a partir de artefatos que envolvem de modelos conceituais da transformação ao teste. Na Tabela 6.5 são apresentados os artefatos relacionados à transformação.

Tabela 6.4 - Principais artefatos relativos a transformação.

Definição	Artefato	Descrição
Arquitetura de Transformação	Requisitos de Transformação	Descreve as necessidades dos arquitetos e dos modelos construídos em um projeto. Também define as funções que devem ser tratadas pela transformação
	Modelo de Regra de Transformação	Define estrutura e comportamento das novas regras de transformação.
	Documentação de Regra de Transformação	Descreve os elementos da estrutura e dos modelos de regras de transformação.
Implementação de Transformação	Implementação de Regra de Transformação	Arquivos pertinentes à implementação das novas regras de transformação conforme a tecnologia escolhida.
	Teste de Regra de Transformação	Teste realizado durante o desenvolvimento ou adaptação de regras de transformação para assegurar a confiança do produto desenvolvido.
	Build de Transformação	Versão operacional da Transformação

	com as novas regras integradas.

Nota-se que os requisitos da transformação vêm do processo de desenvolvimento do sistema, enquanto a transformação integrada é um artefato de saída que retorna não só ao processo, mas à todos outros projetos concorrentes.

6.7.3 Artefatos de Teste

Os artefatos pertinentes a teste continuam os mesmos. A mudança principal é para o caso de teste de modelos. Nesse caso, o artefato de entrada (a ser testado) é a arquitetura do sistema ao invés de uma *build* do sistema. Os documentos de propósito e objetivos dos modelos e transformações e de métricas e guias de avaliação dos modelos também são adições ao processo de teste.

Na Tabela 6.5 são demonstrados os principais artefatos relativos a teste. Alguns artefatos, como relatório de teste, são os mesmos para as três dimensões da disciplina (teste de modelo, de transformações e de software).

Tabela 6.5 - Principais artefatos em relação a teste.

Definição	Artefato	Descrição
Plano de Teste	Propósitos e Objetivos	Descreve e classifica os propósitos e objetivos do modelo ou regra de transformação a ser testado.
	Guia de Métricas e Avaliação	Define métricas e métodos de avaliação conforme os propósitos e objetivos.
	Plano de Teste	Descreve o método de teste e define a forma que os mesmos serão executados.
Teste	Dados de Teste	Descreve os resultados encontrados da execução dos testes. Normalmente qualitativo para modelos e

		quantitativo para *builds*.
	Relatório de Teste	Documento que demonstra as interpretações dos dados de teste. Dá parecer ao andamento e construção do projeto.

6.7.4 Outros artefatos

Outro artefato que se pode destacar no processo de MDE é o Guia de Processo MDE para o projeto. Este artefato é resultado da disciplina de Ambiente. Ele define a adaptação de processo utilizada para o processo específico e principalmente define a forma como o projeto se integra ao desenvolvimento de transformações.

Os outros artefatos trabalhados pelo processo se mantêm similares àqueles encontrados nas definições do RUP. Suas diferenças se dão em sua elaboração, porém não conceitualmente. Sendo assim, essa dissertação não os discute.

6.8 Considerações

Neste capítulo foi apresentado o *framework* de processo criado para desenvolvimento de sistemas em MDE. O processo de baseia em conceitos de desenvolvimento usual encontrados na literatura. Durante a construção do processo, buscou-se aliar as definições do ciclo de desenvolvimento tradicional às boas práticas estudadas para MDE.

As principais diferenças do processo de desenvolvimento em MDE e do processo tradicional estão focadas nas disciplinas centrais de desenvolvimento. A Arquitetura e *Design* passa a abranger uma responsabilidade maior e suas atividades devem ser modificadas para refletir isso. A principal diferença é o foco maior na construção de uma arquitetura rica em informações e estável para uma transformação precisa e confiável.

O processo de MDE também apresenta a adaptação e desenvolvimento de transformações. No processo ela é apresentada como Gerenciamento de Transformações, onde o ciclo de desenvolvimento e adaptação de regras de transformação se integra ao processo de desenvolvimento de software MDE. Essa disciplina acaba substituindo a disciplina de Implementação em um ciclo de desenvolvimento normal.

Outras disciplinas que tem algumas mudanças são Teste e Ambiente. Teste é modificado para abranger o teste de modelos. Como a arquitetura passa a ser o artefato central de desenvolvimento, ela precisam ser testada. As mudanças levam em conta os métodos mais influentes de teste de modelos e validação de transformações encontradas na literatura de qualidade em MDE.

A disciplina de Ambiente passa a apresentar uma preocupação com a adaptação do processo do projeto atual de forma a integrar com o ciclo de desenvolvimento de transformações. Isso é importante, pois o ciclo de desenvolvimento de transformações é relacionado a todos os projetos e deve se integrar a cada um de formas diferentes.

A intenção do processo é conseguir criar uma base geral para desenvolvimento em MDE. Um dos propósitos do paradigma de desenvolvimento em MDE é aumentar a efetividade do processo de desenvolvimento e diminuir os desperdícios.

O processo faz isso ao criar uma transformação integrada. Cada regra que é adaptada ou criada para as equipes acaba entrando em uma contribuição geral. Assim, com o tempo a transformação passa a ser menos desenvolvimento e mais manutenção, permitindo o reuso. O tempo maior utilizado no desenvolvimento da arquitetura é compensado pela automatização do processo de implementação.

É necessário que haja uma avaliação da utilização de MDE para afirmar que isso ocorre. O *framework* teórico e conceitual é ilustrado e avaliado no próximo capítulo.

7 AVALIAÇÃO E ILUSTRAÇÃO DO USO

Neste capítulo se apresenta uma avaliação teórica e ilustrações do uso do *framework* de processo. O objetivo é dar uma confiabilidade conceitual e melhorar a compreensão do processo.

O *framework* apresentado por esta dissertação é de caráter teórico e conceitual. Conforme evidenciou Dwyer (1997), a validação e avaliação prática de *frameworks* é algo que demanda tempo e recursos. Isso acaba dificultando a validação das propostas feitas nos estudos da área. O autor retorna ao tema em Dwyer e Elbaum (2010), afirmando que muitas vezes os dados extraídos de uma avaliação são apenas parciais.

Ericsoon et al. (2010) realizam uma análise teórica e prática de vários *frameworks* utilizados na área de tecnologia da informação. A avaliação teórica pode ser adaptada para o processo aqui apresentado.

A ilustração de uso é utilizada de forma exemplificadora do processo de desenvolvimento apresentado. Fowler (2004) descreve o uso de exemplos para ressaltar os usos e retirar dados e dar especificação de elementos. Além de auxiliar a percepção de melhorias ao processo.

As próximas seções mostram a avaliação do *framework*. Posteriormente é feita uma ilustração sua utilização a partir de um caso documentado.

7.1 Avaliação

A avaliação será dada a partir da adaptação do modelo apresentado por Ericsoon et al. (2010). Os autores apresentam uma série de quinze construtos que o *framework* deve cobrir para ser avaliado positivamente. Na tabela 7.1 apresentam-se os construtos e a avaliação do framework.

Tabela 7.1 - Avaliação do processo segundo os construtos de Ericsoon et al. (2010).

Construto	Descrição	Avaliação
Domínio	Se o processo é genérico ou especializado em uma área de negócios.	Tecnologia da Informação
Base de avaliação	Existe uma estrutura ou documentação para avaliação do processo e dos métodos aplicados.	Sim
Estratégia	Existem definições sobre a direção e futuro (por exemplo, documento de visão, planejamento, etc.)	Sim
Processos e Atividades	Define e estrutura atividades, tarefas ou passos e recomendações de execução.	Sim
Medidas e Métricas	Organiza meios de lidar com objetivos, medidas e métricas para acompanhamento estratégico.	Sim
Melhoria Contínua	Estimula uma melhoria contínua do processo e de seu produto.	Sim
Suporte a Gerência	Estimula liderança e participação dos membros, além de estimular a cooperação dentro da equipe.	Sim
Documentação	O *framework* requer uma documentação geral e específica dos resultados, guias, métodos e do produto.	Sim
Orientação ao cliente	O *framework* promove foco nas necessidades do cliente e busca inserir sua figura no processo.	Sim
Parceria	Integração do processo com parceiros externos ao projeto.	Sim
Responsabilidade	Existe uma distribuição de responsabilidade na organização do processo. Existe comunicação entre as partes responsáveis.	Sim

Papéis	Existe uma definição das posições e funções de cada pessoa dentro do processo.	Sim
Recursos	O *framework* lida com recursos relativos ao processo e sua gerência. Os recursos podem ser físicos ou inerentes ao negócio.	Sim
Recursos Humanos	Lida com a importância de ter as pessoas com habilidades certas nos trabalhos designados.	Sim
Educação, treinamento e ambiente	Existem atividades e guias que estimulem e os membros da equipe fisicamente ou mentalmente. Estimula a cooperação.	Sim

Pelo método de Ericsoon et al. (2010), conceitualmente o *framework* está dando suporte a todos os construtos que os processo de software devem abordar. Ressalta-se que alguns construtos têm suporte, porém, eles devem ser adaptados durante a execução do processo, como medidas e métricas.

O importante é que o *framework* está dando suporte para as atividades de desenvolvimento, de validação e gerenciais em suas definições. Outro ponto que se ressalta é que o processo estimula a colaboração e cooperação entre os papéis. Para os autores, isso é importante para o sucesso das definições de um processo.

7.2 Ilustração de uso

Nesta seção se ilustra o uso do *framework* em uma aplicação. Ressalta-se que a ilustração é conceitual e não prática.

7.2.1 *Tokeneer ID Station*

O *Tokeneer ID Station* foi um projeto desenvolvido pela *Praxis High Integrity Systems* para a NSA (*National Security Agency*, Estados Unidos) para desenvolver parte de um sistema seguro (*Tokeneer System*) de acordo com o processo de desenvolvimento utilizado pela empresa. O projeto foi documentado em TReport (2008) e Moy e Wallenburg (2010).

O processo de desenvolvimento nesse caso foi submetido a seis fases de desenvolvimento: análise de requisitos, especificação formal, projeto, implementação, verificação e teste completo de sistema. A cada fase do projeto foi realizado uma verificação para certificar-se que erros não estavam sendo inseridos no projeto.

O projeto gerou dados e medidas sobre sua execução e o esforço necessário, bem como os custos do projeto. Ao total foram utilizados 260 dias num total de 1736 horas de execução do projeto. Na tabela 7.2 estão demonstrados os dados retirados de TReport (2008).

Tabela 7.2 - Dados do Tokeneer ID Station. Fonte: TReport (2008).

Disciplina/Atividade	Esforço (horas)	Custo (%)
Requisitos	192	11
Especificação do Sistema	234	16
Projeto do Sistema	299	20
Codificação e Verificação	557	34
Teste	76	4
Interface e Integração	316	13
Aceitação	63	2

Pode-se perceber que a maior parte do tempo do projeto foi utilizada em Codificação e Verificação. Existe um grande esforço também na construção das interfaces e da integração do código.

Caso o *framework* e o paradigma fossem utilizados a atividade de Codificação e Verificação seria eliminada. O desenvolvimento de interface passa a integrar o Projeto do Sistema e a Integração passa a ser parte da Gerência de Transformações.

O levantamento de Requisitos e a Especificação do Sistema continuariam ocorrendo conforme ocorreu no projeto. O relatório coloca que o Projeto de Sistema foi realizado em duas partes, uma em que se construiu uma arquitetura abstrata e outra onde se especificou a estrutura. Isso vai de acordo com a especificação do *framework*. Ainda assim, o Projeto de Sistema seria inchado pela utilização de validação de modelos e sua refatoração. Existe também a inclusão dos projetos de interface a serem agregados ao Projeto do Sistema.

As atividades de Teste também seriam acrescidas de esforço pela utilização de teste de modelo e das regras de transformação adaptada. Aumentando as horas que se utilizaria em teste.

A principal adição é a Gerência de Transformação. O tempo e esforço demandado por essa disciplina depende da organização e como que as equipes evoluíram o software de transformação até o momento. No caso, se a empresa Praxis trabalhasse há algum tempo com desenvolvimento MDE, pode-se supor que ela teria mais regras de transformações adaptadas, desenvolvidas e especificadas. Sendo assim, o esforço e tempo demandado pela disciplina seriam pequenos, necessitando somente uma verificação da ferramenta para garantir que ela pudesse lidar com a arquitetura.

Caso a empresa, no entanto, estivesse começando, pode-se supor que o esforço e tempo necessários seriam maiores, devido a maior complexidade e necessidade de implementação de novas regras. Em Hutchinson et al. (2011), os autores expõem que usualmente ao iniciar desenvolvimento MDE, as empresas não tem um ganho muito significativo (apesar de que ele existe). Portanto, supõe-se que as necessidades, nesse caso, aumentariam a demanda de esforço, porém não tanto quanto a Codificação e Verificação.

Qualquer que seja a opção, a organização conseguiria reduzir o tempo de desenvolvimento de seu produto. Por ser uma empresa escolhida para realizar um trabalho para uma agência de suma importância (NSA), acredita-se que o cenário seria o primeiro.

Por fim, o TReport (2008) define os custos do projeto. O custo das disciplinas que seriam retiradas do projeto somam 47% de todo o custo do projeto. A adição e inchamento de outras disciplinas aumentaria o custo. Supõe-se, porém, que assim como o tempo e esforço,

haveria uma diminuição de custos, maior no caso de uma transformação madura, menor no caso da empresa estar iniciando o processo de MDE. A principal contribuição do uso de MDE seria uma melhoria na confiança do código construído e menor desperdício de esforço em funcionalidades nulas.

7.2.2 Dados de Fenton et al.

Fenton et al. (2007) analisam vários projetos de software e quantificam o que normalmente ocorre no desenvolvimento de software comum. Os autores avaliam e colocam o que usualmente ocorre em projetos de software.

Os processos e atividades que normalmente tomam mais tempo e esforços são: (i) especificação e documentação; (ii) projeto e desenvolvimento; (iii) teste e retrabalho; e (iv) gerencia de projetos. Dessas atividades, usualmente o que toma mais tempo (chegando até a metade do tempo total) dos projetos são os pontos ii e iii.

No caso de um processo MDE, essa estrutura seria revisada. O que tomaria mais esforço seria a Gerência de Transformações e o Projeto de Sistema. E os esforços estariam concentrados em criar arquitetura correta para o sistema.

O primeiro passo seria adaptar o processo para as necessidades do projeto. Os autores colocam que maior parte dos projetos tem complexidade média. O maior problema é a construção de soluções sem defeitos.

A adaptação de processo nesse caso já iria levar em conta uma maior atenção à arquitetura a ser construída. O gerente de ambiente também deve perceber que se a complexidade é média e o processo MDE na organização é relativamente novo, pode ser necessário maior tempo e esforço na gerência de transformação, além de um cuidado especial na sua integração.

A parte de retrabalho seria retirada dos projetos. O retrabalho existe em MDE, porém, a refatoração e validação antes da transformação permitem uma maior confiança nos modelos. Além disso, como a criação de código é automatizada, e o processo de transformação é

testado para assegurar precisão, menos defeitos e problemas passam pelo processo, necessitando um menor retrabalho.

Fenton et al. (2007) afirmam que os projetos que mais tem retrabalho e duram mais tempo, usualmente tem custos maiores. A diminuição no tempo de desenvolvimento melhoraria a estrutura de custos. Além disso, a diminuição do retrabalho também diminui o desperdício de recursos e esforços.

Outro ponto ressaltado pelos autores é que o reuso em projetos de software tradicionais muitas vezes acaba criando problemas nas métricas de projeto, além de que os desenvolvedores não podem garantir a qualidade do código portado. Em MDE, o reuso se encontra na automatização e no sistema de transformação legado.

A ideia é que a transformação evolua quanto mais projetos se passam, podendo ser reutilizadas cada vez mais regras, as quais são melhoradas a cada projeto. O *framework* prevê isso dentro da Gerência de Transformações. Como a transformação está sempre passando por testes e avaliações para garantir sua qualidade, o processo de reuso em MDE é mais confiável e estável. Podendo potencialmente aumentar a qualidade dos softwares utilizados.

7.3 Considerações

Neste capítulo apresentou-se uma avaliação do *framework* proposto e ilustrou-se como ele seria usado em uma aplicação real. A pesquisa não teve acesso a recursos ou tempo que permitisse que uma avaliação e estudo prático fossem feitos.

A avaliação conceitual realizada, baseado no modelo dos autores Ericsoon et al. (2010), demonstra que o *framework* vai de acordo com as práticas e dimensões que devem ser abordadas pelo processo de desenvolvimento de software. Isso dá uma avaliação positiva conceitual ao *framework*, mostrando que ele está teoricamente correto e pode representar aquilo que objetiva na prática.

As ilustrações de uso foram utilizadas para demonstrar como o *framework* poderia ser usado na prática. Para isso utilizou-se casos documentados na literatura. No entanto, por não

haver uma aplicação prática, a comparação da ilustração proposta e os dados encontrados pelos autores não é comparável.

Ainda assim, o capítulo permite que o *framework* seja validado conceitualmente. É possível também visualizar a sua aplicação em termos práticos através da ilustração de uso.

Podem-se perceber durante o capítulo as limitações e algumas perspectivas futuras da pesquisa desta dissertação. No próximo capítulo, conclui-se essa dissertação, discutindo as limitações, contribuições e perspectivas futuras desse trabalho.

8 CONCLUSÃO

Model-Driven Engineering é um novo paradigma de desenvolvimento que vem ganhando espaço. A abordagem se torna atrativa pelo fato do conceito trabalhar somente em um nível de abstração mais elevado, tornando o processo de desenvolvimento mais reutilizável, criando um ciclo mais rápido de desenvolvimento e reduzindo custos. No entanto, isso não ocorre sem esforços. Métodos e elementos da engenharia de software devem ser repensados e ajustados para entrar em conformidade com o novo paradigma. A forma como o processo é integrado e as práticas de desenvolvimento precisam ser revisadas.

Este trabalho propôs um *framework* de processo que integre as boas práticas e métodos de garantia de qualidade em MDE. Ao repensar o ciclo de desenvolvimento de software e aliar com os conhecimentos e conceitos de MDE, é possível construir uma estrutura de processo que guie os desenvolvedores de software orientados a modelos.

O *framework* não é definitivo, e nem é o propósito dessa dissertação. O trabalho propõe uma forma de organizar e sequenciar os elementos relacionados ao processo, porém, depende da organização e dos membros da equipe adaptar o processo a sua realidade e necessidades.

A elaboração do *framework* não pode ser realizada sem que exista uma redefinição dos papéis atuantes do processo. Sendo assim, procurou-se definir quais responsabilidades e preocupações os atores devem ter em MDE. O objetivo é guiar os atores dentro das atividades do processo. Ao falar de estrutura e atividades, também é necessário revisar os artefatos que se relacionam a MDE.

Percebeu-se durante o desenvolvimento deste trabalho que MDE gera uma grande mudança nas atividades centrais ao desenvolvimento de software, assim como nos papéis e artefatos trabalhados. Além disso, a integração de todas as partes permitiu uma visão holística de como o processo se estrutura.

Considera-se que a dissertação atingiu o objetivo ao que se propôs. Isso resultou em um *framework* que integra os conceitos de qualidade de software em MDE, dá uma visão

holística do processo de desenvolvimento e procura criar uma sinergia entre os elementos integrantes.

8.1 Contribuições

As contribuições deste trabalho para a evolução do tema foram:

- *Definição de papéis* – identificação dos comportamentos e definição de quais papéis que estão relacionados ao desenvolvimento *model-driven*;
- *Identificação de artefatos* – identificação de quais artefatos são trabalhados durante o processo de MDE;
- *Visão holística* – definir como o desenvolvimento e adaptação de transformações se relaciona com o desenvolvimento do sistema;
- *Levantamento e identificação de melhores práticas* – a identificação das melhores práticas estudadas para qualidade em MDE;
- *Integração de melhores práticas ao processo* – definir a utilização de práticas dentro do processo de MDE e como elas se encaixam no contexto; e
- *Framework de processo* – elaboração e definição de um *framework* de processo para o paradigma *model-driven*.

8.2 Limitações

Existem algumas limitações associadas a esta dissertação. A maior delas é que o *framework* apresentado é teórico e conceitual. Isso não permite que se tenham dados sobre sua aplicação prática ou se consiga refinar ele a partir da aplicação. A falta de validação prática não permite quantificar a efetividade da proposta.

Outra limitação da pesquisa é que ela exige a quem queira aplicar o *framework* que conheça vários aspectos da literatura. Entre eles pode-se afirmar que é necessário

conhecimento de validação de modelos e transformações, além das boas práticas apresentadas.

Uma limitação relacionada à MDE é que o novo paradigma ainda está em evolução. Sendo assim, pode ser difícil para desenvolvedores independentes ou pequenas empresas aplicar o *framework*.

Outra limitação é que o *framework* é genérico e sem refinamento prático para sistemas específicos. A modelagem do processo para uma aplicação pode exigir um grande esforço dos profissionais do projeto.

Por fim, em relação à gerência de projetos também existem limitações. Como discutido durante o trabalho, a mudança de paradigma muda o processo de desenvolvimento e as atividades envolvidas, bem como os perfis das pessoas que trabalham dentro do processo. Logo, os gerentes não tem uma guia de controle e direcionamento de projetos utilizando MDE. Assim, o esforço por parte da gerência para encontrar formas de adaptar as práticas de gerência para MDE é considerado uma limitação.

8.3 Perspectivas futuras

Esta dissertação não pretende finalizar as discussões sobre o tema. As limitações e resultados da pesquisa evidenciam a necessidade de trabalhos futuros dentro do assunto.

Como perspectivas futuras, então, sugere-se realizar uma validação prática do *framework* proposto. Além disso, sugere-se realizar novas avaliações e refinamentos do *framework* buscando uma maior efetividade. Pode-se colocar como sugestão, também, a criação de especificações para adaptação do processo para sistemas específicos.

Outra perspectiva futura é a criação de uma ferramenta que dê suporte ao *framework*. A partir da ferramenta, buscar-se-ia facilitar a adaptação do processo e sua adoção por desenvolvedores com menos recursos financeiros, de tempo ou humanos.

Sugere-se o estudo e a documentação de um guia ou método para integrar a transformação ao processo de desenvolvimento em MDE. O foco seria criar uma integração

eficiente e dar sinergia ao processo. Por fim, sugere-se o estudo e definição de métodos e práticas para a gerência, controle e direcionamento de projetos utilizando o paradigma MDE.

REFERÊNCIAS

ALBERT, M.; CABOT, J.; GÓMEZ, C.; PELECHANO, V. **Automatic Generation of basic schemas from UML class diagrams.** In: Software and Systems Modeling, v.9, n.1, pp.47-67. 2010.

AMELLER, D; FRACH, X. **Considering Non-Functional Requirements in Model-Driven Engineering.** Barcelona: UPC, 93p. Dissertação (Mestrado) -Màster en Computació. Universitat Politècnica de Catalunya. 2009

ANDROMDA. **AndroMDA.** 2012. Disponível em: <www.andromda.org> Acesso em: jan. 2012

ATL. **ATLAS Transformation Language Documentation.** 2011. Disponível em: <http://www.eclipse.org/atl/documentation/> Acesso em: out. 2011

BARBOSA, L.S.; MENG, S. **UML Model Refactoring as Refinement: A Coalgebraic Perspective.** 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. SYNASC'08. 2008

BENCOMO, A. **A step-by-step guide for creating RUP plug-ins.** IBM developerWorks. 2005. Disponível em: <http://www.ibm.com/developerworks/rational/library/05/323_extrup1/extending-the-rup_p1.pdf> Acesso em: abr. 2012

BERENBACH, B.; BOROTTO, G. **Metrics for model driven requirements development.** In: Proceedings of the 28th International Conference on Software Engineering. ICSE'06. 2006

BERGIN, T.J. **History of Programming Languages**. Addison-Wesley Professional. 1st edition. 1996

BOBKOWSKA, A. **Integrating Quality Criteria and Methods of Evaluation for Software Models**. In: Model-Driven Software Development: Integrating Quality Assurance. Edited by Rech and Bunse. 15 pp. Hershey, PA, EUA: Information Science Reference, 2008

BOUKHANOUFA, M-L.; RADERMACHER, A.; TERRIER, F. **Towards a Model-Driven Engineering Approach for developing adaptive real-time embedded systems.** 10th Annual International Conference on New Technologies of Distributed Systems (NOTERE'10). 2010.

BRAGA, C.; MENEZES, R.; COMICIO,T.; SANTOS, C.; LANDIM, E. **Transformation contracts in practice.** In: IET Software, v.6, n.2. 2012.

BROWN, A. **An Introduction to Model-Driven Architecture.** IBM developerWorks, 2004. Disponível em: < http://www.ibm.com/developerworks/rational/library/3100.html> Acesso em: set.2010

CICOZZI, F.; CICCHETTI, A.; SJODIN, M. **Towards a Round-Trip Support for Model-Driven Engineering of Embedded Systems.** 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). 2011

DEMATHIEU, S.; GRIFFIN, C.; SENDALL, S. **Model Transformation with the IBM Model Transformation Framework.** IBM developerWorks. May 2005. Disponível em: <http://www.ibm.com/developerworks/rational/library/05/503_sebas/> Acesso em: abr. 2011

DODIG-CRNKOVIC, G. **Scientific Methods in Computer Science.** In: Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden. 2002

DWYER, M.B. **Automated Analysis of Software Frameworks.** P.paper in ESEC/FSE'97 Worksho on Foundations of Component-Based Systems. September, 1997.

DWYER, M.B.; ELBAUM, S **Unidying Verification and Validation Techniques:** Relating Behavior and Properties through Partial Evidence. In: FoSER'10 Proceedings of the FSE/SDP workshop on Future Software Engineering Research, pp93-98. 2010.

ECLIPSE. **OpenUP – Open Unified Process.** 2011. Disponível em: <epf.eclipse.org/wikis/openup> Acesso em: ago. 2011.

EPF. **Eclipse Process Framework**. 2011. Disponível em: < http://www.eclipse.org/epf/> Acesso em: jul. 2011.

ERICSOON, E; GUSTAFSSON, P.; HÖÖK, D.; MARCKS VON WÜRTEMBERG, L.; ROCHA FLORES, W. **Process Improvement Framework Evaluation.** International Conference on Management Science and Engineering (ICMSE). 2010.

FENTON, N.; NEIL, M.; MARSH, W.; HEARTY, P.; RADLINSKI, L.; KRAUSE, P. **Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction.** 3rd International Workshop on Predictor Models in Software Engineering (PROMISE 2007) part of the 29th Intl. Conference on Software Engineering (ICSE'07). Minneapolis, Estados Unidos, 2007.

FLEUREY, F.; STEEL, J.; BAUDRY, B. **Validation in model-driven engineering: testing model transformations.** In: Proceedings of the First International Workshop on Model, Design and Validation. Nov. 2004.

FOWLER, M. **SpecificationByExample**. 2004. Disponível em: <http://martinfowler.com/bliki/SpecificationByExample.html> Acesso em: abr. 2012

FRANCE, R.; CHOSH, S.; SONG, E.; KIM, D.K. **A metamodeling approach to pattern-based model refactoring.** IEEE Software, v. 20, n. 5, pp 52-58. 2003

GARGANTINI, A; RICCOBENE, E; SCANDURRA, P. **Integrating Formal Methods with Model-Driven Engineering.** Fourth International Conference on Software Engineering Advances. In: Proceedings of ICSEA'09. 2009

GENERO, M.; MIRANDA, D.; PIATTINI, M. **Defining Metrics for UML Statechart Diagrams in a Methodical Way.** In: Proceedings of the 22nd International Conference on Conceptual Modeling. ER2003. LNCS 2814, pp. 118-128. 2003

GENERO, M.; PIATTINI, M.; CRUZ-LEMUS, J.A.; REYNOSO, L. **Metrics for UML Models.** In: CEPIS Upgrade, Vol V, No. 2. April, 2004

GENERO, M. PIATTINI, M.; CALERO, C. **A survey of metrics for UML class diagrams.** In: Journal of Object Technology,v.4, n.9, pp59-92. 2005

HOFSTATER, J. **Model-Driven Development.** Microsoft Developer's Network MSDN. November 2006. Disponível em: <http://msdn.microsoft.com/en-us/library/aa964145.aspx> Acesso em: set. 2010

HOSSEINI, S.; AZGOMI, M.A. **UML Model Refactoring with Emphasis on Behavior Preservation.** Second IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering. TASE'08. 2008

HUMM B.; SCHREIER, U.; SIEDERSLEBEN, J. **Model-Driven Development: Hot Spots in Business Information Systems.** In: European Conference on Model Driven Architecture Fondations and Applications (ECMDA-FA). Nuremberg, Germany, 2005

HUTCHINSON, J.; ROUNCEFIELD, M.; WHITTLE, J. **Model-Driven Engineering Practices in Industy.** In: Proceedings of the 33rd International Conference on Software Engineering. ICSE'11. pp633-642. 2011

HUTCHINSON, J.; ROUNCEFIELD, M.; WHITTLE, J. KRISTOFFERSEN, S. **Empirical Assessment of MDE in Industry.** In: Proceedings of the 33rd International Conference on Software Engineering. ICSE'11. pp471-480. 2011

ISO/IEC 9126. **Software Engineering – Product Quality.** 2001

ISO/IEC 14598. **Information technology -- Software product evaluation**. 1999

JIN, L.; GUISHENG, Y. **Method of constructing model transformation rule based on reusable pattern.** International Conference on Computer Application and System Modeling. ICCASM. 2010.

JUNG, C.F. **Metodologia para pesquisa e desenvolvimento.** Rio de Janeiro: Axcel Books. 2004.

KLEPPE, A. **MCC: A Model Transformation Environment.** European Conference on Model Driven Architecture Fondations and Applications. ECMDA-FA'06. Bilbao, Espanha. 2006

KEPPLE, A.; WARMER, J.; BAST, W. **MDA Explained, The Model Driven Architecture: Practice and Promise.** Addison-Wesley, 2003

KHÜNE, T. **What is a Model?** In: Dagstuhl Seminar Proceedings. 2005

KÜHNE, T.; ATKINSON, C. **Model-Driven Development: A Metamodeling Foundation**. In: IEEE Software, vol.20, pp36-41, 2003

KONRAD, S.; GOLDSBY, H.J.; CHENG, B.H.C. **iMAP: An Incremental and Iterative Modeling and Analysis Process.** In: International Conference on Model Driven Engineering Languages and Systems (MoDELS). Nashville, EUA. 2007

KOSICANSKI, A.; SOARES, M.S. **Qualidade de Software.** 2ª edição. São Paulo: Novatec. 2007

LANGE, C.F.J.; CHAUDRON, M.R.V. **Managing Model Quality in UML-based Software Development.** In: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05). 2005

MA, H.; SHAO, W.; ZHANG, L.; MA, Z.; JIANG, Y. **Applying OO metrics to assess UML meta-models.** In: Proceedings of the Seventh Internation Conference on the Unified Modeling Language (UML'2004). LNCS 3273, pp. 54-68.October 2004

MA, H.; SHAO, W.; ZHANG, L.; MA, Z.; JIANG, Y. **Towards the UML evaluation using taxonomic patterns on meta-classes.** In: Proceedings of the Fifth International Conference on Quality Software. QSIC'05. pp37-44. 2005

MAHMOOD, S.; LAI, R. **Measuring the Complexity of a UML Component Specification.** In: Proceedings of the Fifth International Conference on Quality Software. QSIC'05. 2005

MCNEILE, A. **MDA: The Vision with the Hole.** 2003. Disponível em: <http://www.metamaxim.com/download/documents/MDAv1.pdf> Acesso em: dez. 2011

MELLOR, S; BALCER, M. **Executable UML: A Foundation for Model-Driven Architecture.** 2nd edition. Addison Wesley. 2002

MELLOR, S.; CLARK, A.; FUTAGAMI, T. **Model-Driven Development.** In: IEEE Software, vol. 20, pp. 14-18. 2003

MELLOR, S; SCOTT, K.; UHL, A.; WEISE, D. **MDA Distilled, Principles of Model Driven Architecture.** Addison-Wesley Professional, 2004

MENS, T.; TAENTZER, G.; MUELLER, D. **Model-Driven Software Refactoring.** In: Model-Driven Software Development: Integrating Quality Assurance. Edited by Rech and Bunse. 33 pp. Hershey, PA, EUA: Information Science Reference, 2008

MOHAGHEGHI, P.; AAGEDAL, J. **Evaluating Quality in Model-Driven Enginnering.** In: International Workshop on Modeling in Software Engineering (MISE'07). 2007

MOHAGHEGHI, P.; DEHLEN, V.. **Existing Model Metrics and Relations to Model Quality.** ICSE Workshop on Software Quality. WoSQ '09. 2009

MOHAGHEGHI, P.; DEHLEN, V.; NEPLE, T. **Definitions and Approaches to Model Quality in Model-Based Software Development – A Review of Literature.** In: Proceeding of Information & Software Technology. 1646-1669. 2009

MOLA. **MOdeling LAnguages**. 2012. Disponível em: < http://modeling-languages.com/> Acesso em: mar. 2012

MONPERRUS, M.; JÉZÉQUEL, J-M.; CHAMPEAU, J.; HOELTZENER, B. **Measuring Models.** In: Model-Driven Software Development: Integrating Quality Assurance. Edited by Rech and Bunse. 20 pp. Hershey, PA, EUA: Information Science Reference, 2008

MOY, Y.; WALLENBURG, A. **Tokeneer: Beyond Formal Program Verification**. Presented at Embedded Real Time Software and Systems. May, 2010.

NBR ISO/IEC 9126-1. **Engenharia de software - Qualidade de produto. Parte 1: Modelo de qualidade. Associação Brasileira de Normas Técnicas**. 2003

NUGROHO, A.; CHAUDRON, M. **Managing the Quality of UML Models in Practice.** In: Model-Driven Software Development: Integrating Quality Assurance. Edited by Rech and Bunse. 36 pp. Hershey, PA, EUA: Information Science Reference, 2008

OMG. **MDA Guide (Version 1.0.1)**. Object Management Group. 2001. Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf> Acesso em: jul. 2010

OMG. **Meta Object Facility (MOF) Core Specification (Version 2.4.1).** Object Management Group. 2011. Disponível em: < http://www.omg.org/spec/MOF/2.4.1> Acesso em: mar. 2012

PÉREZ-MARTÍNEZ, J. E.; SIERRA-ALONSO, A. **From Analysis Model to Software Architecture: A PIM2PIM Mapping.** European Conference on Model-Driven Architecture Fondations and Applications. ECMDA-FA'06. Bilbao, Espanha. 2006

PRESSMAN, R.S. **Engenharia de Software.** São Paulo: McGraw Hill. 2006

PUNTER, T.; VOETEN, J.; HUANG, J. **Quality in Model Driven Engineering.** In: Model-Driven Software Development: Integrating Quality Assurance. Edited by Rech and Bunse. 19pp. Hershey, PA, EUA: Information Science Reference, 2008

RAHIMI, S.K.; LANO, K. **Integrating Goal-Oriented Measurement for Evaluation of Model Transformation.** International Symposium on Computer Science and Software Engineering (CSSE). 2011

RECH, J.; BUNSE, C. **Model-Driven Software Development: Integrating Quality Assurance.** Hersey, PA, EUA: Information Science Reference. 2008

RSA. **Rational Software Architect.** IBM. 2012. Disponível em: < http://www.ibm.com/developerworks/rational/products/rsa/> Acesso em: jan. 2012

SANTOS, J.P.; MOREIRA, A.; ARAÚJO, J; GOULÃO, M. **Increasing Quality in Scenario Modeling with Model-Driven Development**. Seventh International Conference on the Quality of Information and Communications Technology. In: Proceedings of QUATIC'10. 2010

SANTOS, L.B.; PRETZ, E. **Framework para Especialização de Modelos de Qualidade de Produtos de Software.**. 15p. Trabalho de Conclusão (Pós-Graduação) Pós-Graduação em Qualidade de Software. Centro Universitário Feevale. Novo Hamburgo: Feevale, 2010

SCHMIDT, D.C. **Model-Driven Engineering**. In: IEEE Computer, (Vol. 39, No. 2) pp. 25-31. February 2006

SEIDEWITZ, E. **What Models Mean.** In: IEEE Software, vol. 20, pp26-32. 2003

SELIC, B. **The Pragmatics of Model-Driven Development**. IEE Software, vol. 20, no. 5, pp. 19-25. 2003

SHIROMA, Y.; WASHIZAKI, H.; FUKAZAWA, Y.; KUBO, A.; YOSHIOKA, N. **Model-Driven Security Patterns Apllication Based on Dependences among Patterns.** International Conference on Availability, Reliability, and Security. ARES'10. 2010

SHUJA, A.K.; KREBS, J. **IBM Rational Unified Process Reference and Certification Guide**. IBM Press. First edition. 2008

SPEM. **Software & Systems Process Engineering Metamodel Specification 2.0**. OMG: 2008. Disponível em: <http://www.omg.org/spec/SPEM/> Acesso em: mai. 2011

STERRITT, A.; CAHILL, V. **Customisable Model Transformations Based on Non-Functional Requirements**. 2008 IEEE Congress on Services. SERVICES 2008: Part I. 2008.

STERRIT, R.; HINCHEY, M. **Organic Computing and Model-Driven Engineering in Embedded Systems.** IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. ISORC'09. 2009

TREPORT. **Tokeneer ID Station – EAL 5 Demonstrator: Summary Report.** 2012 Disponível em: <http://www.adacore.com/uploads/downloads/Tokeneer_Report.pdf> Acesso em: ago. 2011.

VANHOOFF, B.; AYED, D.; BAELEN, S. V.; JOOSEN, W.; BERBERS, Y. **UniTI: A Unified Transformation Infrastructure**. International Conference on Model Driven Engineering Languages and Systems (MoDELS). Nashville, TN, Estados Unidos. 2007

VELOCITY. **The Apache Velocity Project.** 2011. Disponível em: <http://velocity.apache.org/index.html> Acesso em: nov. 2011

VIATRA2. **VIsual Automated model Transformation Documentation.** 2011. Disponível em: <http://www.eclipse.org/gmt/VIATRA2/doc/> Acesso em: out. 2011

WANGER, S; DEISSENBOECK, F. **An Integrated Approach to Quality Modeling**. Fifth International Workshop on Software Quality. In: Proceedings of ICSE'07, p.6. 2007

WHOLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M.C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering.** Estados Unidos: Kluwer Academic Publishers. 2000

YIN, R.K. **Estudo de Caso – Planejamento e Métodos.** 2ª edição. Porto Alegre: Bookman. 2001