DOI: 10.5748/9788599693100-11CONTECSI/PS-996

ADDING QUALITY IN SOFTWARE WITH A PROPOSAL FOR INTEGRATION OF MODELS DRIVEN SOFTWARE DEVELOPMENT AND TESTING

Carla C. de Jesus Almeida (Universidade Federal de Sergipe, Sergipe, Brasil) - ccjalmeida@gmail.com

Adicinéia A. de Oliveira (Universidade Federal de Sergipe, Sergipe, Brasil) - adicineia@ufs.br

Software engineering has as its objective the development of software products that meet customer needs. For this, there are new approaches to development and testing of software, such as Model-Driven Development (MDD) and Model Driven Testing (MDT), which aim to shift the focus of development and software testing for models and their transformations, and no longer in programming languages. This work aims to propose a process model for model driven software development, allowing the tests accompany every stage of this, adding quality to the developed software.

Keywords: Model Process; Model-Driven Development (MDD); Model Driven Testing (MDT).

ADICIONANDO QUALIDADE NOS SOFTWARE COM UMA PROPOSTA PARA INTEGRAÇÃO DO DESENVOLVIMENTO E TESTE DE SOFTWARE DIRIGIDO A MODELOS

A Engenharia de Software possui como objetivo o desenvolvimento de produtos de software que atendam às necessidades do cliente. Para isso, surgem novas abordagens de desenvolvimento e teste de software, tais como: o Desenvolvimento Dirigido por Modelos (MDD) e o Teste Dirigido por Modelos (MDT), as quais têm como objetivo mudar o foco do desenvolvimento e do teste de software para os modelos e suas transformações, e não mais em linguagens de programação. Este trabalho tem como objetivo propor um modelo de processo de desenvolvimento de software dirigido por modelo, permitindo que os testes acompanhem todas as etapas deste, agregando qualidade ao software desenvolvido.

Palavras-chave: Modelo de Processos; Desenvolvimento Dirigido a Modelos (MDD); Testes Dirigido a Modelos (MDT).

1. Introdução

A experiência inicial na construção de sistemas de *software* mostrou que o desenvolvimento informal não era suficiente, onde, projetos importantes de desenvolvimento de *software* atrasavam, tinham custos que superavam as previsões,

possuíam baixa qualidade e não tinham desempenho satisfatório, além de serem difíceis de manter. O resultado apontou para a necessidade de criação de novas técnicas e métodos para controlar a complexidade inerente aos grandes sistemas de *software* (SOMMERVILLE, 2007).

Ao longo do tempo, houve uma evolução dos conceitos, modelos, processos e técnicas na área de Engenharia de *Software*, porém, alguns problemas ainda persistem, sendo eles: (i) portabilidade, devido a atualização ou modificação frequentemente da tecnologia; (ii) interoperabilidade, visto que, existe uma dificuldade na comunicação entre os sistemas; (iii) manutenção e documentação, pelo fato, da documentação não ter nenhuma influência no código e vice-versa; e, (iv) produtividade, devido os desenvolvedores muitas vezes considerarem que codificar seja mais produtivo que elaborar modelos e documentação (NIKULSINS & NIKIFOROVA, 2008), dificultando com isso o desenvolvimento de *software* com qualidade.

Para desenvolver *software* com qualidade é importante à utilização de um processo de desenvolvimento, o qual deve funcionar como uma estrutura de tarefas necessárias para construção de um *software*. Porém, a existência de um processo de desenvolvimento de *software* não é garantia de que o *software* terá qualidade e atenderá as necessidades dos clientes (PRESSMAN, 2010). Desta forma, com um processo formal de teste de *software*, é possível agregar qualidade ao *software* desenvolvido, como também, realizar melhorias no processo de *software* utilizado, através do aperfeiçoamento de suas atividades (SANTOS, 2006).

Como contribuição a este cenário, pesquisadores estão investigando o uso de modelos como apoio as atividades tanto de desenvolvimento quanto de teste de *software* (JAVED, STROOPER, & WATSON, 2007). Estas abordagens são denominadas como dirigidas por modelos. Para Booch (1999), os modelos auxiliam na visualização de como é o sistema, ou como queremos que ele seja; permitem especificar a estrutura e o comportamento do sistema; oferecem auxílio na construção de sistemas; e, documentam a tomada de decisões.

A disciplina de Engenharia de *Software* em que os modelos são os artefatos centrais do desenvolvimento, isto é, são utilizados na construção do sistema, na comunicação das decisões de *design,* etc. é chamada de Engenharia Dirigida por Modelos, em inglês, *Model Driven Engineering* (MDE) (MILICEV, 2009). Outras abordagens têm sido propostas nos últimos anos, dentre elas: o *Model Driven Development* (MDD), *Model Driven Architecture* (MDA) e *Model Driven Testing* (MDT), nas quais os modelos são criados usando conceitos que são menos ligados à tecnologia de implementação subjacente e são mais próximo do domínio do problema em relação às linguagens de programação (SELIC, 2008).

Este artigo tem como objetivo propor um modelo de processo de desenvolvimento de *software* dirigido por modelo, visando melhorar o desenvolvimento de sistemas de *software* a partir da uniformidade com o processo de teste de *software* dirigido por modelos, permitindo que os testes acompanhem todas as etapas deste, agregando qualidade ao *software* desenvolvido.

Segundo Wazlawick (2008), a pesquisa realizada nesta proposta é classificada em relação a apresentação de uma forma diferente de resolver um problema, visando buscar uma solução para a problemática abordada através de um mecanismo diferente dos estudados

e/ou propostos anteriormente, de maneira a alcançar resultados melhores dentro de um determinado contexto.

O artigo está estruturado da seguinte forma. A Seção 2 apresenta sobre os processos de desenvolvimento e de testes de *software.* Na seção 3 é apresentada uma visão geral sobre as abordagens dirigidas por modelos. A Seção 4 apresenta uma análise sobre o enfoque dos testes de *software* nas abordagens dirigidas por modelos. Na Seção 5, um modelo de integração para o desenvolvimento e teste de *software* dirigido por modelos é apresentado. Na Seção 6, é feita uma análise em relação aos trabalhos relacionados. Por fim, na Seção 7, são discorridas as considerações finais.

2. Processos de Desenvolvimento e Teste de *Software*

É perceptível como as organizações estão aumentando sua dependência tecnológica, levando suas operações internas a serem conduzidas e direcionadas por um conjunto cada vez maior de sistemas informatizados, buscando reduzir custos e a ampliar sua forma de atuação no mercado através do uso da tecnologia. Desta forma, todas as variáveis envolvidas no processo de desenvolvimento de *software* têm um nível crescente de complexidade, ocasionando riscos de mal funcionamento, os quais aumentam proporcionalmente à complexidade do ambiente, tornando-se mais difícil desenvolver *software* com um nível de qualidade aceitável (BARTIÉ, 2002).

Para France e Rumpe (2007), um fator significativo por trás da dificuldade do desenvolvimento de *software* é a grande diferença conceitual entre o problema e os domínios de implementação. Essa distância entre o cenário de desenvolvimento de *software* e o cenário idealizado junto à Engenharia de *Software*, ocorre devido ao mal uso e até mesmo o não uso dos fundamentos da Engenharia de *Software* para apoiar as atividades do desenvolvimento, tendo como consequência, o crescente custo com a manutenção dos *software*, gerando retrabalho em nível de requisitos, projeto, codificação e teste, atribuído a uma mal definição do domínio do problema nas fases iniciais do desenvolvimento (ÁVILA & SPÍNOLA, 2007).

Mohan, Shankar e Jayasridevi (2012) mostram que as atividades de concepção do *software* introduzem de 50% a 65% de defeitos durante o processo de *software*. Processos de teste e de revisão precisam ser estabelecidos durante as atividades do processo de desenvolvimento para reduzir substancialmente a grande porcentagem de defeitos, reduzindo o custo das atividades subsequentes do processo de *software*. A tabela 01 mostra a porcentagem de defeitos introduzidos em cada fase do ciclo de vida de desenvolvimento de *software*.

Tabela 01: Defeitos introduzidos em diferentes fases do desenvolvimento do *software.*

Fase do Desenvolvimento de *Software*	**Percentual de Defeitos Introduzidos**
Requisitos	20%
Projeto (*Design)*	25%
Codificação	35%
Manuais de Usuário	12%

Correções ruins	8%

Fonte: Traduzido de Mohan, Shankar e Jayasridevi (2012).

Apesar do avanço no desenvolvimento de *software* nos últimos 40 anos, muitas empresas ainda estão presas a antigos paradigmas, impedindo o seu amadurecimento no processo de desenvolvimento de *software* (BARTIÉ, 2002), visto que, grande parte dos produtos de *software* ainda são entregues depois do prazo, acima do orçamento previsto, sem atender às necessidades dos clientes e com baixa confiabilidade devido à falta de qualidade (SCHACH, 2008).

Para minimizar esta situação, a Engenharia de *Software* tem contribuído através de ferramentas de apoio para as atividades, métodos para orientar a realização de atividades, processos para definir as atividades e produtos, bem como, garantir a qualidade tanto do processo quanto do produto de *software*. Nesse contexto, processos de *software* são importantes, visto que, estabelecem quais atividades que permitem a obtenção do produto de *software* com qualidade (HIRAMA, 2012). Na próxima seção serão abordados sobre alguns modelos de processo de desenvolvimento de *software*.

2.1 Processos de Desenvolvimento de *Software*

Para Kleppe, Warmer e Bast (2003), o processo de desenvolvimento de *software* é muitas vezes impulsionado pelo baixo nível de *design*, onde os documentos e diagramas produzidos durante as fases iniciais, como as descrições de requisitos e os vários diagramas em UML, perdem rapidamente o seu valor, logo após o início da fase de codificação, devido as alterações feitas ao longo do tempo, ocasionando uma distância entre o código e a documentação.

Os modelos de processo de *software* servem como uma definição de alto nível das fases que ocorrem durante o desenvolvimento. Eles não têm como objetivo proporcionar definições detalhadas, mas, destacam as principais atividades e suas interdependências (SWEBOK, 2004). Na seção 2.2.1 é apresentada uma revisão teórica sobre os modelos de processo de *software*.

2.2.1 Modelos Prescritivos de Processo

Os modelos prescritivos de processo foram originalmente propostos para organizar o desenvolvimento de *software*, os quais têm trazido uma estrutura útil para o trabalho de Engenharia de *Software*. Além disso, têm fornecido um roteiro efetivo para equipes de *software*, indicando um conjunto de elementos de processo composto por: tarefas, produtos de trabalho, mecanismos para garantia de qualidade e controle de mudanças para cada projeto de desenvolvimento de *software* e um fluxo de trabalho, o qual define como os elementos do processo interagem. Os principais modelos de processos de desenvolvimento de *software* são: Modelo em Cascata, Evolucionário e Incremental (PRESSMAN, 2010), os quais são brevemente descritos a seguir.

2.2.1.1 Modelo em Cascata

O Modelo em Cascata sugere uma abordagem sistemática e sequencial para o desenvolvimento de *software*, iniciando com as especificações do cliente que progride ao longo do planejamento pelas fases de modelagem, construção e implantação, culminando na manutenção progressiva do *software* acabado (PRESSMAN, 2010). Inicialmente, o resultado de cada fase é formado de um ou mais documentos aprovados, sendo que a fase seguinte não deve iniciar antes que a fase anterior tenha terminado (SOMMERVILLE,

2007). A Figura 01 apresenta as fases do modelo em cascata, sendo que na primeira fase, ocorre a iniciação do projeto e a especificação dos requisitos do *software*. As estimativas de prazo, custo, como também, a elaboração do cronograma do projeto são feitas na fase de Planejamento. Na fase de Modelagem, a análise e o projeto do sistema são realizados. Na fase de construção, o *software* é efetivamente desenvolvido e os testes são realizados e, por fim, na fase de implantação o *software* é entregue, sendo que, após a entrega, o *software* passa por manutenções.

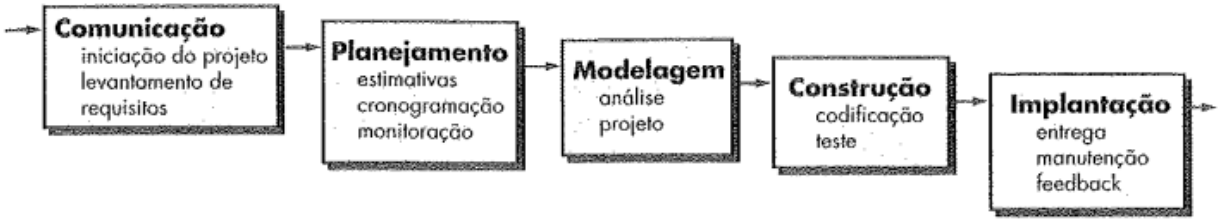

Figura 01: O modelo em cascata.
Fonte: Pressman (2010).

Atualmente, os produtos de *software* são desenvolvidos rapidamente e sofrem muitas modificações, para estes casos, o Modelo em Cascata é inadequado. No entanto, pode servir como um modelo de processo útil para casos nos quais os requisitos são fixos e o trabalho segue até o fim de forma linear (PRESSMAN, 2010).

2.2.1.2 Modelo Evolucionário

Os Modelos Evolucionários são baseados na ideia de desenvolvimento de uma implementação inicial, a qual mostra o resultado ao cliente por meio de versões sucessivas e incrementais, até que o sistema seja desenvolvido por completo (SOMMERVILLE, 2007).

O Modelo Espiral é um exemplo de Modelo Evolucionário que combina a natureza iterativa da prototipagem com os aspectos controlados e sistemáticos do modelo em cascata. Usando o modelo espiral, o *software* é desenvolvido em uma série de versões evolucionárias. Durante as primeiras iterações, as versões podem ser um modelo de papel ou protótipo. Nas últimas iterações são produzidas versões cada vez mais completas do sistema submetido à engenharia (PREESMAN, 2010). A figura 02 apresenta o funcionamento do modelo, onde o primeiro círculo em torno da espiral pode resultar no desenvolvimento da especificação de um produto, e passagens subsequentes em torno da espiral podem representar o desenvolvimento de um protótipo e progressivamente, versões mais sofisticadas do *software*.

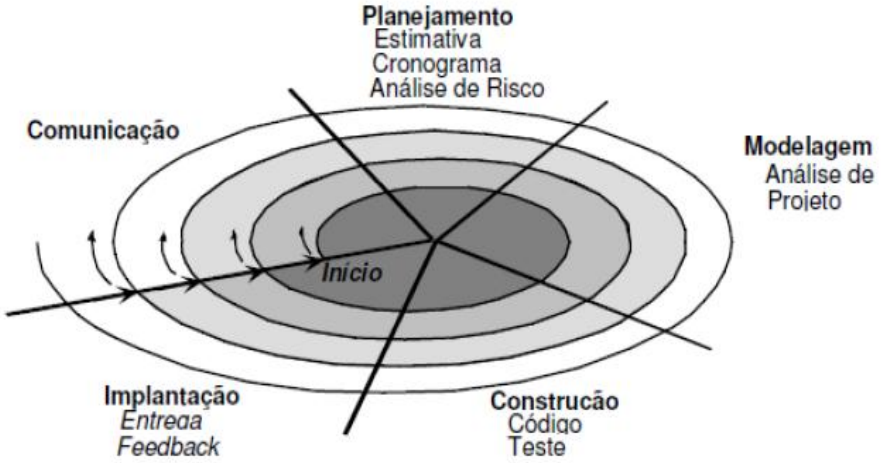

Figura 02: Um modelo espiral típico.
Fonte: Pressman (2010).

De acordo com Pressman (2010), a abordagem evolucionária exige uma competência considerável tanto na análise de riscos, quanto para aplicá-la com êxito, visto que, caso um risco importante não seja identificado e mitigado, fatalmente ocorrerão problemas.

2.2.1.3 Modelo Incremental

O Modelo Incremental combina elementos do modelo em cascata aplicando-os de maneira iterativa, conforme ilustrado na figura 03. Cada sequencia linear produz "incrementos" do *software* passíveis de serem entregues. Quando um Modelo Incremental é usado, o primeiro incremento frequentemente é chamado de núcleo do produto, onde, os requisitos básicos são atendidos. O núcleo do produto é usado pelo cliente e um plano é desenvolvido para o próximo incremento como resultado do uso e/ou avaliação (PRESSMAN, 2010).

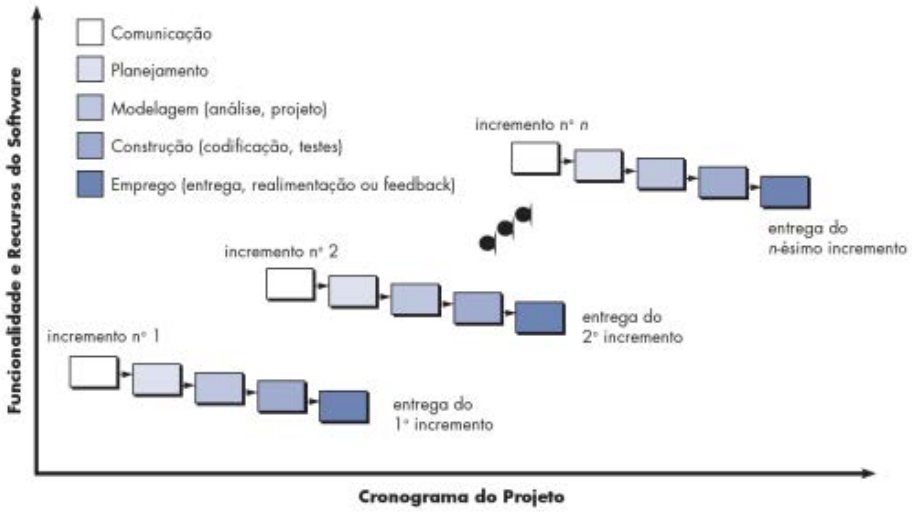

Figura 03: O modelo incremental.
Fonte: Pressman (2010).

Vasconcelos (2006) cita as principais vantagens do modelo incremental, são elas: (i) a funcionalidade do sistema estará disponível mais cedo, pois ela é entregue a partir dos

incrementos; (ii) os incrementos iniciais agem como um protótipo para auxiliar no levantamento dos requisitos para os incrementos finais; (iii) diminuição dos riscos de falhas no projeto como um todo; e, (iv) os serviços de prioridade mais alta do sistema tendem a receber mais testes.

2.2.2 Modelo V

O modelo V é um aprimoramento do modelo cascata, onde as atividades construtivas são separadas das atividades de testes (IEEE/IEC 12207, 2008 *apud* SPILLNER, LINZ & SCHAEFER, 2011). O modelo tem a forma de um "V", onde as atividades de construção, desde a definição de requisitos até a implementação, são encontrados no ramo descendente do "V". As atividades de teste ficam no ramo ascendente e são ordenadas em níveis de teste, e combinadas para o nível de abstração apropriado sobre a atividade do lado oposto da construção, conforme mostra a figura 04 (SPILLNER, LINZ & SCHAEFER, 2011).

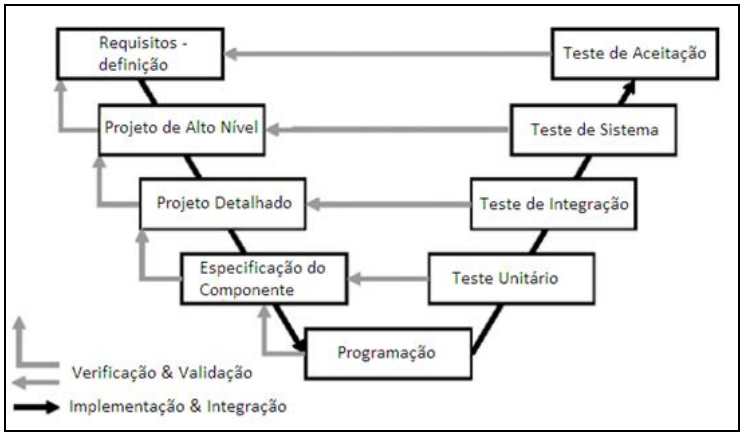

Figura 04: Modelo V Geral.
Fonte: Spillner, Linz e Schaefer (2011).

Além disso, o modelo V retrata a importância do teste do *software* no início do desenvolvimento do ciclo e garante a qualidade do *software*, visto que, é possível executar testes várias vezes ao longo do ciclo. Em geral, esse modelo reforça a ideia de que o teste não é uma fase, mas uma parte integrante do ciclo de desenvolvimento do *software* (SPILLNER, LINZ & SCHAEFER, 2011).

Bastos *et al.* (2007) cita que o modelo V é uma representação gráfica de um exemplo que representa a integração do desenvolvimento com o teste de *software*, tendo em vista sua simplicidade em apresentar as atividades em linhas gerais.

Desta forma, percebe-se que o teste é uma parte integral de vários modelos de desenvolvimento de *software*, tais como: sequencial (modelo em cascata, modelo V e modelo W), iterativo (*Rapid Application Development* - RAD e modelo em espiral) e incremental (evolucionário e metodologias ágeis) (ISTQB, 2007). Na próxima seção, serão apresentados conceitos relacionados ao Teste de *Software,* bem como, alguns processos de testes e o alinhamento das atividades de teste com os modelos de desenvolvimento de *software.*

2.3 Teste de *Software*

Myers (2004) define teste de *software* como um processo de execução de um programa com a intenção de encontrar erros. Além disso, pode ser considerado também, como uma

das técnicas de V&V mais utilizadas, consistindo de um processo dinâmico que envolve a execução do *software* com dados de teste, com o intuito de verificar se o seu comportamento está de acordo com o que foi especificado pelo cliente, ou seja, o objetivo principal é encontrar o maior número de defeitos possíveis (MACIEL, 2010).

Segundo o SWEBOK (2004), teste de *software* é uma disciplina de Engenharia de *Software* que trata da avaliação da qualidade do produto e da identificação de defeitos e problemas, bem como, consiste na verificação do comportamento de um programa em um conjunto finito de casos de teste.

Embora frequentemente aponte que o teste de *software* é uma fase no processo de desenvolvimento de *software*, isto é, como se o mesmo fosse apenas uma atividade distinta que ocorre em um ponto particular no processo de desenvolvimento, não se deve confundir essa etapa intensa de testes com o processo de teste e análise como um todo que deve ser feita ao longo do processo de desenvolvimento de *software* (PEZZÈ & YOUNG, 2008).

Sousa *et al.* (2011), afirmam que o teste deve ser continuamente mantido e implementado ao longo de todo o ciclo de vida de um *software*, sendo possível reduzir a probabilidade de ocorrência de defeitos, com a aplicação de um processo de teste de *software* integrado ao processo de desenvolvimento de *software* garantindo o atendimento as necessidades dos clientes e minimizando riscos de negócio e de Tecnologia da Informação (TI). Portanto, a execução do teste deve ser feita com um processo bem definido e integrado. Na seção 2.3.1 serão apresentados alguns processos de teste que orientam sobre as principais atividades.

2.3.1 Processo de Teste de *Software*

Na visão de Engenharia de *Software,* um processo é definido como um "conjunto de passos parcialmente ordenados, cujo objetivo é atingir uma meta para entregar um produto de *software* de maneira eficiente, previsível e que atenda as necessidades do negócio" (MOLINARI, 2008).

Para Bath e Mckay (2011) o processo de teste de *software* consiste de algumas atividades individuais, a saber:

- Planejamento e Controle: o planejamento é a atividade que define os objetivos do teste e especifica as atividades de teste, a fim de cumprir com os objetivos, os recursos humanos que serão alocados no projeto de teste. A atividade de controle é contínua, ou seja, visa monitorar as atividades de teste durante todo o projeto, comparando o progresso real com o que foi planejado.
- Análise e *Design*: onde os objetivos gerais de teste são transformados em condições de teste tangíveis e em casos de teste.
- Implementação e Execução: os procedimentos de teste ou *scripts* são especificados pela combinação dos casos de teste em uma determinada ordem e incluindo qualquer outra informação necessária para a execução do teste, o ambiente é preparado e os testes são executados.
- Avaliação dos Critérios de Saída e Relatórios: a execução do teste é avaliada em função dos objetivos definidos (ISTQB, 2010).
- Fechamento das Atividades de Teste: os dados das atividades do teste (artefatos gerados – *testware*, fatos e números) são coletados para avaliar a experiência. As atividades de encerramento de teste ocorrem em etapas do projeto, tais como quando um sistema de *software* é lançado, ou quando um

projeto de teste ou um marco é concluído (ou cancelado), ou ainda quando uma versão de manutenção foi concluída (ISTQB, 2010).

A figura 05 apresenta os cinco passos genéricos que essas atividades geralmente se encaixam:

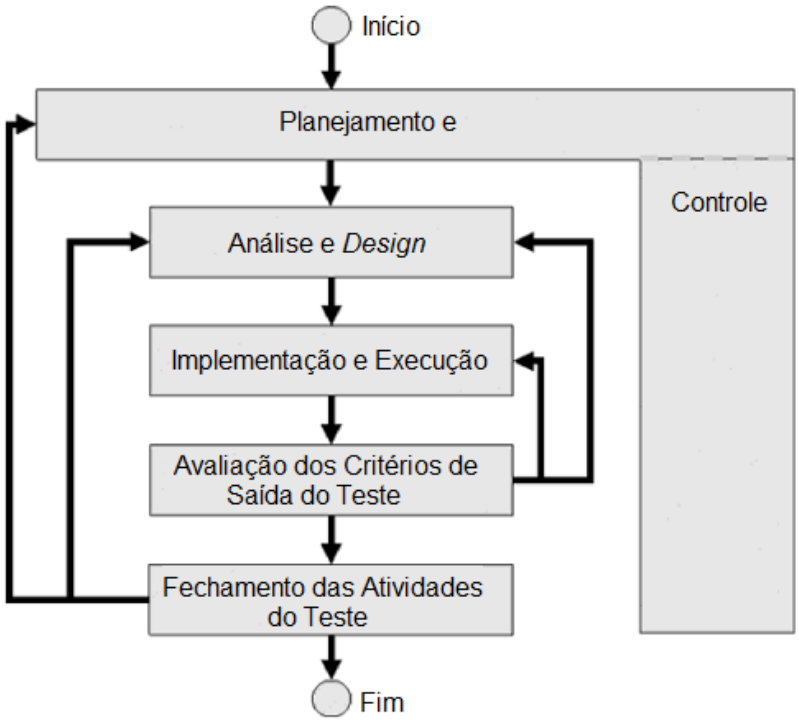

Figura 05: O Processo de Teste Fundamental.
Fonte: Traduzido de Bath e Mckay (2011).

O processo de teste de *software*, apresentado na figura 06, é baseado na metodologia *Test Management Approach* (TMap), composto por quatro fases sequenciais e duas paralelas. Na fase de procedimentos iniciais é realizado um estudo dos requisitos do negócio que darão origem ao *software* a ser desenvolvido. Na fase de planejamento a estratégia de testes e o plano de testes são elaborados. Enquanto isso, o ambiente de testes (equipamentos, ferramentas, treinamento, etc.) é organizado na fase de preparação. Na fase de especificação são elaborados os casos e roteiros de testes que servirão de base para os testes na fase de execução. Por fim, na fase de entrega, documentação final é elaborada, relatando as ocorrências relevantes à melhoria do processo (RIOS & MOREIRA, 2003 *apud* BASTOS *et al*., 2007).

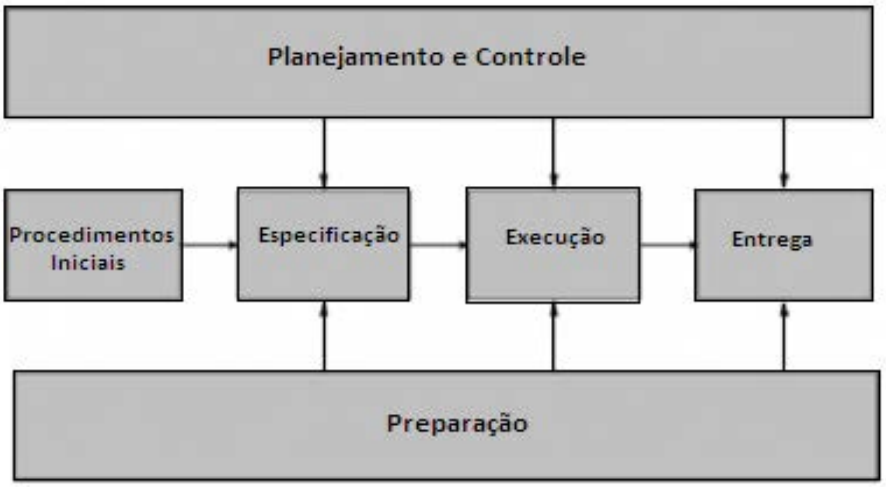

Figura 06: Modelo 3P X 3E do ciclo de vida do processo de teste.
Fonte: Rios e Moreira (2003) *apud* Bastos *et al*. (2007).

O Processo de Teste de *Software* (PTS) do Departamento de Informática do Sistema Único de Saúde (DATASUS) é organizado em atividades baseadas em artefatos de entrada, controles e recursos e que geram artefatos de saída. As atividades são realizadas por perfis responsáveis e representadas através de diagramas de atividades, seguindo a mesma abordagem do RUP (DATASUS, 2008). A figura 07 apresenta a interface do PTS.

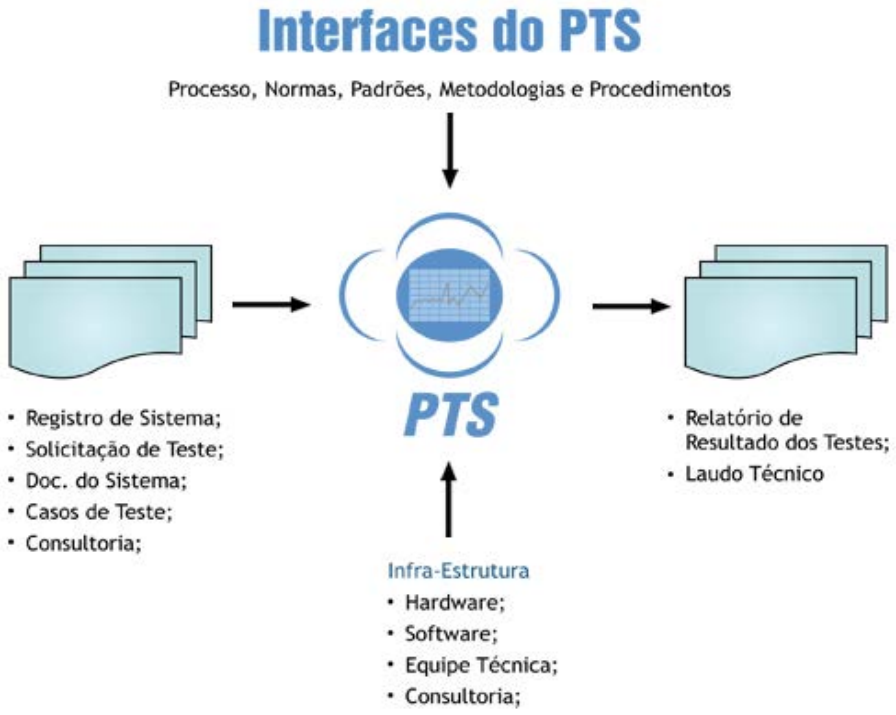

Figura 07: Interface do Processo de Teste de *Software* do DATASUS.
Fonte: DATASUS (2008).

A figura 08 mostra que o PTS do DATASUS tem início com uma demanda de teste realizada por um agente público do Ministério da Saúde, essas demandas são cadastradas no SCP. A partir do cadastro é instanciado o fluxo de trabalho do processo de teste, onde na fase de planejamento identifica-se o escopo do projeto de teste. É iniciado também o

controle do fluxo de trabalho e o acompanhamento dos requisitos de teste. Após o planejamento, é iniciada a fase de acompanhamento e monitoramento das incidências com o intuito de garantir a rastreabilidade da situação das incidências até o seu fechamento. Os cenários e os casos de testes são projetados de acordo com os requisitos de teste, como também, a preparação do ambiente de teste através da instalação e configuração dos recursos computacionais necessários para apoiar a execução do teste. Para formalizar o encerramento do projeto de teste é realizada uma reunião de encerramento do projeto com a participação do solicitante para entrega do relatório de resultado dos testes (DATASUS, 2008).

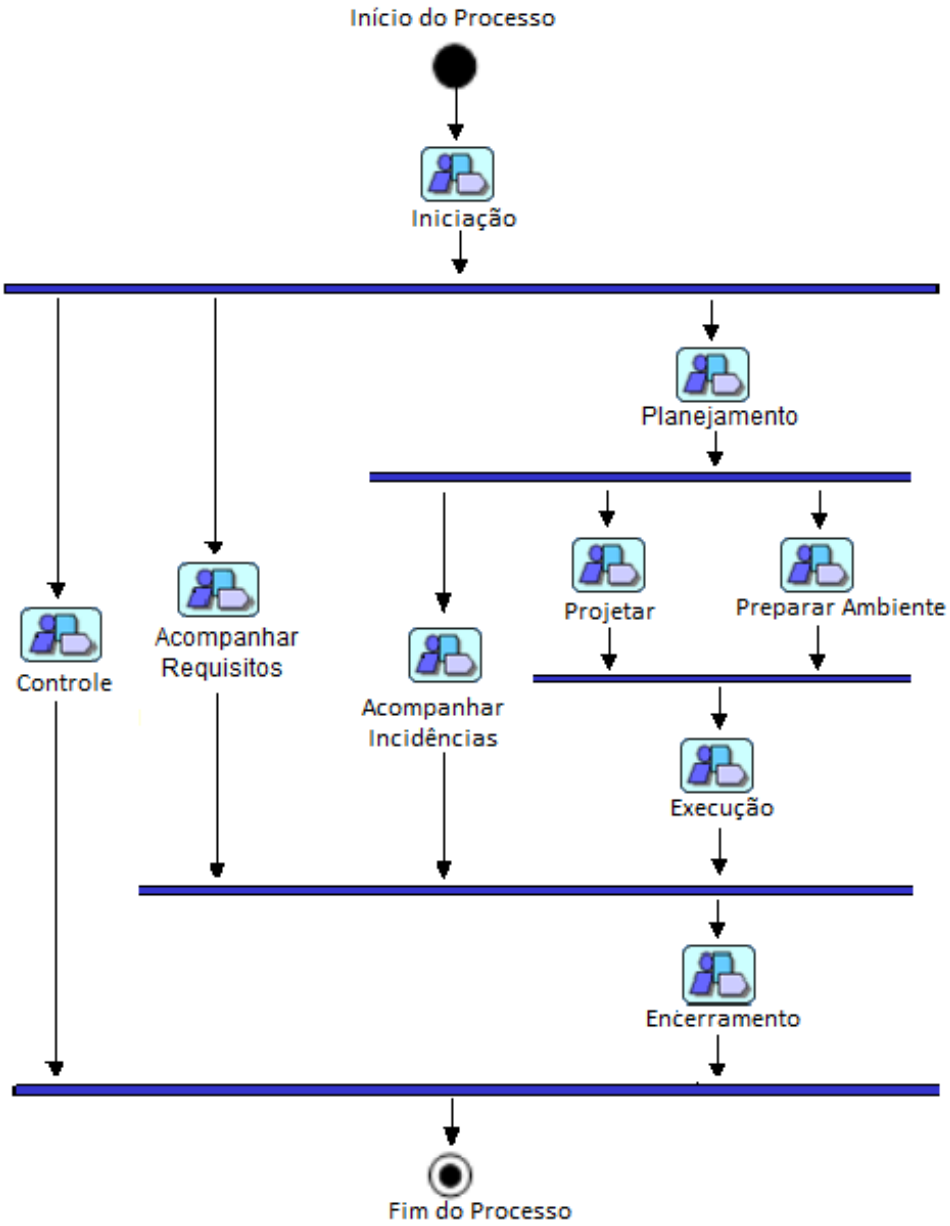

Figura 08: Fluxo de Atividades do Processo de Teste de *Software* do DATASUS.
Fonte: DATASUS (2008).

Pádua (2003) *apud* Santos (2006) apresenta um fluxo de teste (Cf. Figura 09) com dois grupos de atividades para cada conjunto de testes, (conhecido também como bateria de testes) a ser desenvolvido: preparação e realização. O planejamento da bateria é elaborado durante a preparação, juntamente com as especificações de cada teste. Durante a realização, os testes são executados com o intuito de encontrar os erros, para que se possível sejam corrigidos e os relatórios de teste sejam elaborados. A preparação e realização de cada bateria de teste correspondem a uma execução completa do fluxo de teste. O grupo de preparação compreende as atividades de Planejamento, na qual o plano de testes é elaborado, e *Design*, a qual produz as respectivas especificações. O grupo de realização é formado pelas demais atividades: Implementação, que monta o ambiente de testes; Execução, que os executa efetivamente, produzindo os principais relatórios de testes; Verificação do término, que determina se a bateria pode ser considerada como completa; e o Balanço Final, que analisa os resultados da bateria, produzindo um relatório de resumo.

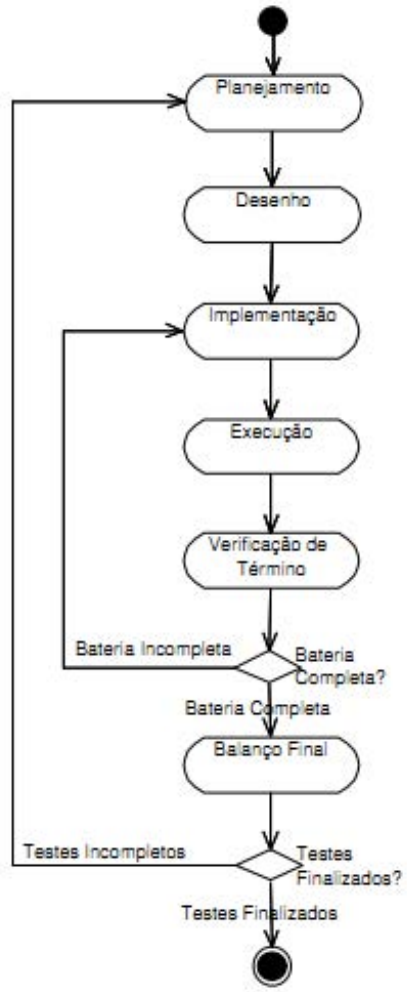

Figura 09: O Fluxo de teste.

Fonte: Pádua (2003) *apud* Santos (2006).

O processo de teste deve ser revisto continuamente, de forma a realizar melhorias e possibilitar aos profissionais envolvidos uma maior visibilidade e organização de suas atividades, resultando numa maior agilidade e controle operacional do projeto de teste

(BARTIÉ, 2007). Além disso, ao final é necessária uma avaliação uma avaliação crítica das atividades executadas, levando em consideração os recursos gastos e os resultados alcançados, com o intuito de promover melhorias a serem aplicadas em projetos de testes posteriores, alcançando uma melhoria contínua do processo (SPILLNER, 2011). Na próxima será apresentada uma análise sobre o teste de *software* nos modelos de desenvolvimento de *software* apresentados.

2.4 Análise do Teste nos Modelos de Desenvolvimento de *Software*

Em todos os modelos de processo, o *software* é desenvolvido em fases, sendo que a maior parte deles possuem fases semelhantes, diferindo principalmente nas condições e possibilidades de progressão para a fase seguinte ou na revisão de uma fase anterior (BAKER *et al*., 2008).

Segundo o ISTQB (2012a), as atividades de teste devem ser alinhadas ao modelo de processo de *software* selecionado, cuja natureza pode ser sequencial, iterativo ou incremental. O Quadro 01 apresenta uma análise sobre o teste de *software* nos modelos de processo de *software* apresentados neste trabalho.

Nos modelos sequenciais, como o cascata, V e W, todos os produtos de trabalho e atividades de uma determinada fase são concluídos antes do início da próxima fase (ISTQB, 2012b). Porém, no modelo cascata após a conclusão da fase de implementação e início da fase de teste, aparecem todas as inconsistências do levantamento de requisitos, deficiências de análise e modelagem, como também, os erros da implementação do *software.* Esta concentração de problemas numa única fase torna-se o processo de detecção e correção extremamente complexo, caro e de alto risco (BARTIÉ, 2002).

No modelo V, o processo fundamental de teste do ISTQB® aplicado ao nível de teste de sistema poderia ser alinhado da seguinte forma:

- O planejamento do teste deve ocorrer simultaneamente com o planejamento do projeto, e o controle do teste continuar até as atividades de encerramento do projeto;
- A análise e modelagem do teste ocorrerem concorrentemente com a especificação de requisitos, modelagem de sistema e arquitetura (alto nível) e com a especificação da modelagem de componente (baixo nível);
- A preparação do ambiente de teste deve iniciar durante a modelagem do sistema;
- A execução do teste ter início quando o critério de entrada do teste for atendido (ou abandonado), o que normalmente significa que pelo menos tanto o teste de componente, quanto o de integração de componentes estejam completos. É importante que a execução do teste continue até que o critério de saída do teste tenha sido satisfeito.

Já os modelos iterativo ou incremental, tais como o RAD e o *Rational Unified Process* (RUP), não seguem a mesma ordem de atividades do modelo V, visto que, é possível excluir algumas destas, por exemplo, no iterativo é possível fazer uso de um processo de teste reduzido, normalizado para cada iteração, onde as fases de análise, execução, avaliação dos resultados dos testes e a elaboração dos relatórios podem ser realizadas para cada iteração, enquanto que a fase de planejamento é feita no início do projeto e os

relatórios no final (ISTQB, 2012a). Para cada nova iteração é necessário um novo ciclo de testes, onde novos cenários de testes vão sendo incorporados e armazenados para que sejam reaproveitados em outras iterações (BARTIÉ, 2002).

No modelo espiral, os protótipos gerados são testados para determinar quais aspectos dos problemas técnicos permanecem sem solução. Após a resolução dos principais problemas técnicos, o projeto prossegue de acordo com qualquer modelo sequencial ou interativo (ISTQB, 2012b).

Quadro 01: Análise sobre o teste de *software* nos modelos de processo de *software*.

Modelo de Processo de *Software*	**Teste de *Software***	
	Pontos Fortes	**Pontos Fracos**
Cascata	Método rigoroso dirigido por documentação, auxiliando na execução dos testes a partir de um melhor entendimento da equipe de teste na regra de negócio do *software*.	Erros nos requisitos e projeto somente podem ser encontrados na fase de teste, ou seja, ao final do processo, onde é mais caro para corrigi-los. O teste é entendido como uma ação única e ao final do projeto, imediatamente antes da liberação, fazendo uma analogia a uma inspeção de fabricação antes de entregar o produto ao cliente.
Evolucionário/ Espiral	É possível executar testes em cada iteração, permitindo que os defeitos sejam encontrados mais cedo (MOLINARI, 2008).	Risco dos defeitos ocorrerem pode continuar se propagando caso os testes forem insuficientes.
Incremental	Permite a execução de testes nas entregas parciais do desenvolvimento; É possível executar testes primeiro em áreas importantes.	Tempo para testar e fazer testes de regressão são elevados.
Modelo V	Integração com os testes ao longo do ciclo de vida do *software*, visto que, as atividades de teste são iniciadas em paralelo com as de desenvolvimento; O teste é contínuo, tendo um custo benefício em termos de defeitos encontrados, bem como, a facilidade para corrigi-los.	Pode haver baixa integração entre as fases devido ao isolamento destas, podendo implicar um mau aproveitamento dos recursos disponíveis no projeto (MOLINARI, 2008); Um teste é concebido a partir de um único documento, sem ser modificado por documentos posteriores ou anteriores; Necessita de uma grande interação entre todas as fases do ciclo. Muitos recursos são necessários para testar todos os derivados do projeto.

Fonte: Adaptado de Evans (2004) e Schach (2008).

De acordo com Spillner, Linz e Schaefer (2011) a descrição das tarefas de teste nos modelos de desenvolvimento apresentados, não é suficiente como uma orientação sobre como realizar testes estruturados em projetos de *software*. Para a incorporação do teste no processo de desenvolvimento, é necessário um processo mais detalhado, onde o conteúdo da fase seja dividido em tarefas menores, iniciando a partir do planejamento do teste, em seguida, projetando os casos de teste, preparando-se para a execução e avaliando o estado até o fechamento do teste.

3. Abordagens Dirigidas por Modelos

A evolução de abordagens dirigidas por modelos apresentam novas formas de, não apenas gerenciar a complexidade do desenvolvimento de *software*, mas também em criar casos de teste, automatizar procedimentos de teste, além de incentivar a reutilização, possibilitando assim reduzir a quantidade de erros e acelerar a produção de *software* (SOUSA *et al.,* 2011). Além disso, estes tipos de abordagens oferecem diversos benefícios, dentre eles: redução de custos e tempo de desenvolvimento de *software*, melhoria da qualidade dos produtos de *software*, aumento no retorno sobre os investimentos em tecnologia e rápida inclusão de benefícios de tecnologias emergentes nos sistemas existentes (OMG, 2013).

Segundo Kent (2003), em uma abordagem orientada a modelo, a visão é que os modelos se tornam artefatos para serem mantidos junto com o código. A figura 10 apresenta uma representação das abordagens dirigidas por modelos.

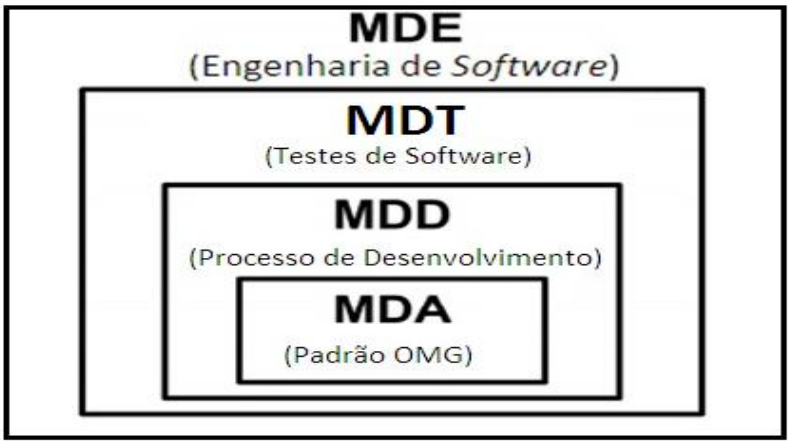

Figura 10. Representação das iniciativas dirigidas por modelos.
Fonte: Adaptado de Ameller (2009).

De acordo com Ameller (2009) a MDE é um superconjunto do MDD (Cf. Figura 1), pelo fato de englobar outras tarefas dirigidas por modelos do processo completo da Engenharia de *Software*, por exemplo, a evolução de sistema dirigido por modelo. Já a MDA é uma visão particular do *Object Management Group* (OMG), sendo um *framework* conceitual que realiza a abordagem do MDD, a qual visa estabelecer um conjunto de modelos para apoiar a construção de *software* para que eles sejam os principais artefatos do processo de desenvolvimento (MAGALHÃES *et al.*, 2011). Como a MDA depende da utilização de padrões OMG, ela pode ser considerada como um subconjunto do MDD, sendo que este último contribui principalmente pela flexibilidade para definição dos processos de desenvolvimento de *software* (FOWLER, 2005). Já o MDT visa agregar qualidade aos produtos de *software* que são desenvolvidos seguindo estas abordagens. Nas próximas seções as abordagens dirigidas por modelos são apresentadas.

3.1 MDE

A MDE tem como objetivo organizar os níveis de abstração e metodologias, motivando o uso de modelos para descrever tanto o problema quanto a sua solução em diferentes níveis de abstração. Além disso, gerencia e manipula os modelos para a criação de sistemas de *software* com qualidade e baixo custo para atender às necessidades do usuário (FONDEMENT & SILAGHI, 2004).

De acordo com Díez e Uriarte (2004), a MDE não implica numa mudança radical no processo de desenvolvimento de *software*, isto é, as atividades de um projeto de *software* são semelhantes, sendo elas: elicitação de requisitos, análise de *software*, arquitetura e *design* detalhado, implementação, teste e instalação. As principais diferenças com as abordagens tradicionais de desenvolvimento de *software* são: (i) os modelos são os elementos-chave do desenvolvimento de *software;* (ii) o código é derivado a partir dos modelos bem definidos e completos; e, (iii) os relacionamentos, rastreabilidade, derivação e transformações entre modelos diferentes são automatizados, tanto quanto possível.

Outra abordagem de desenvolvimento é o MDD, a qual visa oferecer uma maior produtividade sem perda de qualidade aos produtos de *software* desenvolvidos. O MDD é apresentado a seguir.

3.2 MDD

O MDD tem como objetivo mudar o foco do desenvolvimento de *software* para os modelos e transformações entre modelos, e não mais em linguagens de programação, contribuindo para aumentar a qualidade do *software*, bem como facilitar a portabilidade e diminuir os custos inerentes à evolução. Com essa mudança de foco, a equipe de desenvolvimento concentrará seus esforços na construção de modelos mais completos e precisos, elevando o nível de abstração e viabilizando a total geração de código de forma automática (ALVES, MACHADO & RAMALHO, 2008).

A principal ideia no MDD é a transformação de modelos de maiores níveis de abstração (domínio do problema) em modelos mais concretos (domínio solução) até se obter, por fim, o código do sistema. O paradigma MDD preconiza que o desenvolvimento inicial e as modificações futuras da aplicação sejam efetuados apenas no modelo mais abstrato (BUARQUE, 2009).

A OMG propôs a realização do MDD em MDA, a qual tem como características ser uma abordagem padronizada aberta e independente de fornecedor (KLEPPE *et al.,* 2003; GHOLAMI & RAMSIN, 2010). A MDA será abordada na próxima seção.

3.3 MDA

A MDA segue a filosofia da MDE, mudando o foco das atividades do desenvolvimento para os modelos e transformações que levam à geração de código. Em contraste com os modelos de processos tradicionais de desenvolvimento, um processo de MDA requer a seleção de metamodelos e regras de mapeamento que transforma modelos em código de aplicação (DÍEZ & URIARTE, 2004).

A OMG prevê que tecnologias como a MDA irá fornecer os meios para integrar mais facilmente as novas infraestruturas de implementação em projetos existentes, bem como,

gerar porções significativas de código específico de aplicação, arquivos de configuração, pontes de integração de dados e outros artefatos de implementação de infraestrutura a partir de modelos. Além disso, irá sincronizar melhor a evolução dos modelos e suas implementações conforme o progresso do *software*, e rigorosamente simular e testar os modelos (FRANCE & RUMPE, 2007).

A Figura 11 ilustra como os padrões do OMG se encaixam em MDA, onde o núcleo da arquitetura, no centro da figura, é baseado em padrões de modelagem OMG: *Unified Modeling Language* (UML), *Meta Object Facility* (MOF) e o *Common Warehouse Metamodel* (CWM), que compreende um número de perfis UML, os quais representam as características comuns de todas as plataformas de *middleware* apropriadas para a sua categoria de computação, mas, que será independente de qualquer plataforma específica. Além disso, os *Plataform Independent Models* (PIM) podem ser realizados através da MDA em praticamente qualquer plataforma, aberta ou proprietária, incluindo *Web Services,*.NET, CORBA, J2EE, entre outras (OMG, 2013).

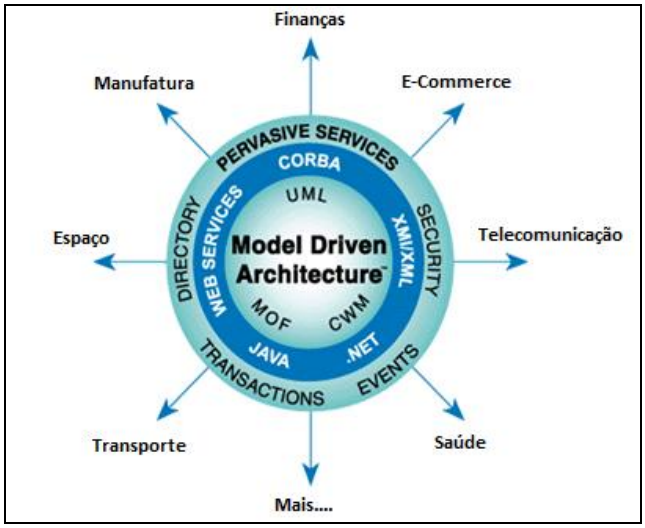

Figura 11. Arquitetura Dirigida por Modelo da OMG.
Fonte: Traduzido de OMG (2013).

Os principais componentes de uma abordagem MDA são: (i) Metamodelos, estruturas que descrevem como modelos devem ser formados; (ii) Modelos, instâncias dos metamodelos; e (iii) Transformações, regras que definem como modelos de entrada devem ser transformados em modelos de saída, com base em seus metamodelos (MACIEL, MACHADO & RAMALHO, 2009).

Para agregar qualidade aos produtos de *software* desenvolvidos com as abordagens dirigidas por modelos, têm-se o MDT, o qual será apresentado na próxima seção.

3.4 MDT

MDT é uma abordagem representante da MDE tornando-se promissora para automação do teste de *software*, tendo como objetivo reduzir os esforços na criação de casos de teste de *software*, através da geração automática destes a partir dos modelos, permitindo um desenvolvimento de *software* com os padrões de qualidade (IBM, 2003).

Segundo Lima *et al.* (2007), ao fazer uso da MDE com foco em testes, são esperados dois benefícios, sendo que o primeiro é a redução do custo de implantação de um determinado

componente e os testadores de correspondentes em múltiplas plataformas através da reutilização dos modelos. O segundo é sobre alcançar um maior nível de automação no processo de geração de casos de teste, através da especificação de transformações entre os modelos. Além disso, os modelos de teste podem auxiliar a descobrir erros antes da transformação dos modelos em código (FRANCE *et al.*, 2006).

A fim de obter benefícios em relação a separação do PIM e *Plataform Specific Models* (PSM) na geração e execução de testes, é necessário que a estratégia de MDT refine-se a três tarefas clássicas: (i) a geração de casos de teste a partir de modelos de acordo com um dado critério de cobertura; (ii) a geração de um oráculo de teste para determinar os resultados esperados de um teste; e, (iii) a execução de testes em ambientes de teste, possivelmente também gerados a partir de modelos. Sendo que a primeira e segunda tarefa são independentes de plataforma, ao contrário da terceira, a qual deve ter uma certa plataforma (HECKEL & LOHMANN, 2003).

Após o estudo dos conceitos relacionados as abordagens dirigidas por modelos, foi analisado como o teste de *sofware* é executado em alguns processos e metodologias que fazem uso destas. Na próxima seção uma análise é apresentada.

4. Análise sobre o Teste de *Software* nas Abordagens Dirigidas por Modelos

O Quadro 02 apresenta uma análise sobre o teste de *software* nas abordagens de desenvolvimento de *software* dirigidas por modelos, sendo que na maioria destas, o teste de *software* é executado como uma atividade.

Quadro 02. Análise sobre o teste de *software* nas abordagens dirigidas por modelos.

Processo/Metodologia	Tipo de Abordagem	Inclui Teste de *Software*?
C³	MDD	Parcialmente
SDL-MDD	MDD	Não
MODA-TEL	MDA	Parcialmente
MASTER	MDA	Parcialmente
MDDP	MDA	Parcialmente

O C³ é um processo de desenvolvimento de *software* que faz uso de técnicas de MDD e tem como objetivo o aumento da produtividade, qualidade e flexibilidade no desenvolvimento de aplicações de negócios. Compreende atividades de verificações de consistência e testes de integração de componentes na fase de desenvolvimento do *software* (HILDENBRAND & KORTHAUS, 2004).

SDL-MDD possui como foco principal o desenvolvimento de sistemas distribuídos e sistemas de comunicação na área de computação ubíqua. Além disso, é suportado por uma suíte de ferramentas e usa uma estrutura de sistema genérica para obter tanto a separação limpa quanto a reusabilidade dos modelos independentes de plataforma e dos modelos específicos de plataformas (KUHN *et al.*, 2006).

A metodologia MODA-TEL fornece a identificação das fases e atividades individuais e suas inter-relações em um processo de desenvolvimento de *software* dirigido por modelo. (GAVRAS *et al.*, 2003). Nessa metodologia, o teste de *software* é abordado somente na fase de execução, sendo realizado manualmente de acordo com os vários níveis de teste.

MASTER é uma metodologia de desenvolvimento de *software* com foco em MDA, que descreve tanto o processo do MDD quanto o workflow do processo principal (LARRUCEA, DÍEZ & MANSELL, 2004). O teste é uma das fases do processo e tem como objetivo mostrar que os requisitos estão atendidos.

O MDDP está em conformidade com a MDA (CRAG SYSTEMS, s.d). Entretanto não apresenta claramente as classificações do CIM, PIM e PSM em seus processos: incremental e cascata, conforme orienta a OMG (2003). Ambos preveem o teste como uma atividade em seus fluxos, sugerindo um fluxo de tarefas de testes, não apresentando um processo efetivo dirigido por modelos para realização do teste de *software*.

Nota-se que a realização de testes de *software* nas abordagens acima apresentadas é incipiente, isto é, as que preveem sua execução, não mostram uma metodologia ou um processo que oriente a implantação e/ou realização em conjunto com o desenvolvimento de *software* dirigido por modelos. Para uma realização integrada das atividades destas abordagens é proposto neste trabalho, um modelo de processo de desenvolvimento de *software,* o qual é apresentado na seção a seguir.

5. Modelo de Desenvolvimento de *Software Qualitas*

As principais funções de um modelo de processo de *software* são para determinar a ordem das etapas envolvidas no desenvolvimento, a evolução do *software*, bem como, estabelecer os critérios de transição de uma fase para a próxima. Assim, um modelo de processo aborda as seguintes questões de projeto de *software*: (i) O que vamos fazer a seguir?; e (ii) Por quanto tempo vamos continuar a fazê-lo? (BOEHM, 1988).

O modelo *Qualitas* está estruturado no modelo espiral e organizado em fases, onde cada volta na espiral representa uma fase do processo de *software* e sua execução é feita evolucionariamente ou incrementalmente. Cada ciclo envolve uma progressão que aborda uma sequência de passos para cada protótipo do produto e a dimensão angular representa o progresso alcançado na conclusão de cada ciclo da espiral. A Figura 12 apresenta o modelo de processo de desenvolvimento de *software* Qualitas.

O modelo de Processo de *Software* Orientado a Modelos proposto por este trabalho apresenta um alinhamento das principais atividades do processo de desenvolvimento de *software* (1ª parte) com a integração das atividades de um processo de MDD (2ª parte) com um de processo de MDT (3ª parte).

O modelo é iniciado com a criação do Plano de Requisitos, no qual as diretrizes para elaboração dos documentos de requisitos são descritas. Além dos tipos de requisitos e de seus atributos, como também, a sua rastreabilidade para melhor gerenciá-los. Em seguida é elaborado o Plano de Ciclo de Vida e feita a Revisão de ambos os planos.

A partir da primeira fase, continua com a criação do Modelo de Negócios com o intuito de estabelecer uma melhor compreensão entre o negócio da organização e o sistema de *software* que será desenvolvido. Posteriormente, define-se o CIM, o qual apresenta exatamente o que o sistema deve fazer, porém, ocultando todas as especificações relacionadas com a TI, mantendo-se independente de como o sistema vai ser ou é atualmente implementado (TRUYEN, 2006). Em seguida, os *Computational Independent Testing Models* (CITM) são gerados, os quais são responsáveis por capturar os objetivos de teste. Após, a análise de requisitos é iniciada, fase em que os requisitos são elicitados para

a elaboração do PIM, consequentemente gera-se os *Plataform Independent Testing Models* (PITM), que são modelos que refletem a arquitetura de teste e os casos de teste abstratos.

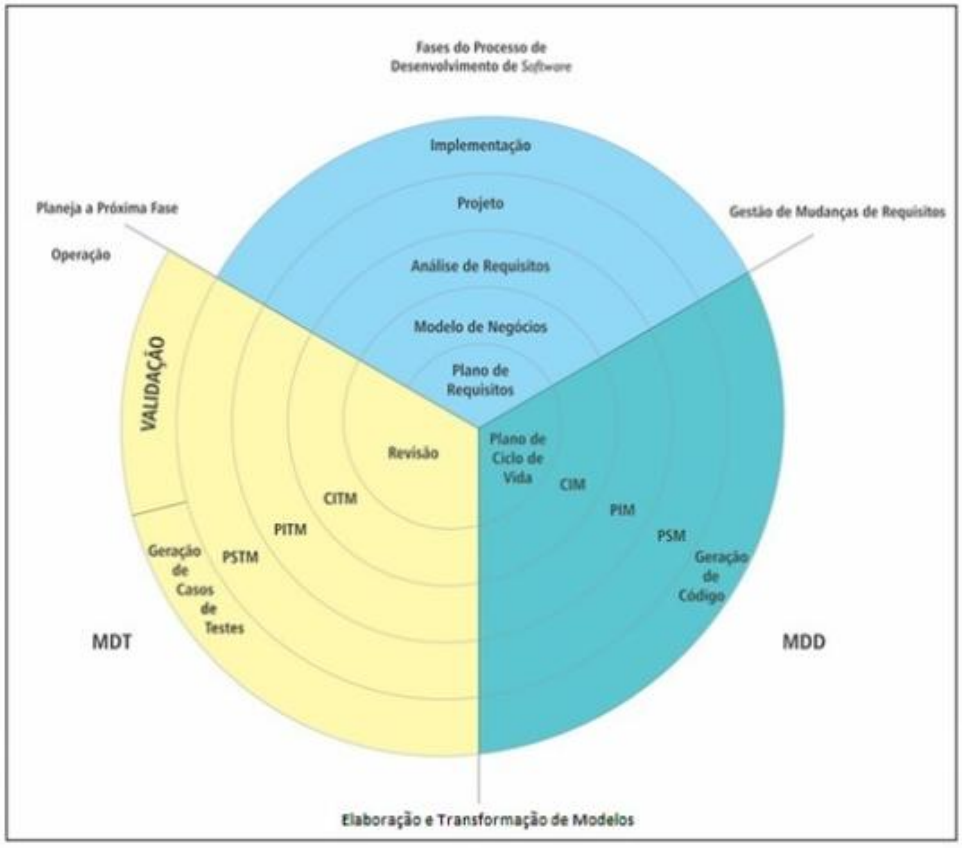

Figura 12: Modelo de Processo de *Software* Orientado a Modelos.

A fase de projeto tem como objetivo mostrar como o sistema vai ser concretizado, com isso um ou vários PSM são criados, bem como, os *Plataform Specific Testing Model*s (PSTM), ao quais são modelos de testes específicos para uma determinada plataforma. Na fase de implementação, o ambiente é criado tanto para que o código seja gerado a partir dos PSMs quanto os casos de testes a partir dos PSTMs para que a fase de validação seja realizada, com o intuito de validar se a versão do *software* atende as necessidades do cliente. Caso esteja atendendo, o mesmo é disponibilizado para uso, ou seja, entra em operação. Caso contrário, as correções são feitas e validadas com o re-teste ou teste de regressão.

O ciclo de vida do modelo de processo é constante, onde o *software* é desenvolvido numa série de versões incrementais. Cada ciclo inclui três atividades principais:

- Gestão de Mudanças de Requisitos: faz o controle e acompanhamento dos requisitos individuais, bem como, mantém as relações entre os requisitos dependentes, de forma que seja possível avaliar o impacto das mudanças dos requisitos;
- Elaboração e Transformação de Modelos: os modelos de desenvolvimento são criados e transformados, os quais servem de base para os modelos de teste. Além disso, são também mantidos a partir das alterações nos requisitos.
- Planejamento da Próxima Fase: o projeto é analisado visando à tomada de decisão quanto ao prosseguimento para o próximo *loop* da espiral. Caso seja

decidido prosseguir, são elaborados planos para a próxima fase, caso contrário, elabora-se o plano de encerramento do projeto.

Segundo Mellor *et al.* (2005) para tirar um maior proveito da atividade de elaboração de modelos, é importante seguir alguns princípios: não elaborar um modelo que não seja necessário e não elaborar modelos que não foram feitos para serem expostos.

O Quadro 03 apresenta as atividades que estão contidas em cada etapa da estrutura do modelo *Qualitas*.

Quadro 03: Detalhamento das atividades do modelo *Qualitas*.

Fases	Etapas	Atividades
Fase 01	CIM	Identificar e Descrever os Processos *As-Is*
		Modelar os Processos *As-Is* Identificados
		Revisar os Artefatos *As-Is* gerados
		Homologar dos Artefatos *As-Is* Revisados
		Redesenhar os Processos
		Apresentar os Artefatos *To Be*
		Modelar os Modelos Independentes de Computação
		Selecionar a Estratégia das Transformações
	CITM	Gerar os Modelos de Testes Independentes de Computação
		Testar os Modelos Independentes de Computação
		Analisar os Resultados dos Testes
Fase 02	PIM	Gerar o Modelo Independente de Plataforma
		Selecionar a Plataforma Específica para a Geração do(s) PSM(s)
	PITM	Gerar os Modelos de Testes Independentes de Plataforma
		Testar os Modelos Independentes de Plataforma
		Analisar os Resultados dos Testes
Fase 03	PSM	Gerar os Modelos Específicos de Plataforma
		Criar as Transformações Adicionais e *Scripts* de Geração de Código para Plataformas Alvo
	PSTM	Gerar os Modelos de Testes Específicos de Plataforma
		Testar os Modelos Específicos de Plataforma
		Analisar os Resultados dos Testes
Fase 04	Geração de Código	Gerar o Código Fonte a partir do PSM
		Implementar se necessário o Código-Fonte
	Geração de Casos de Testes	Gerar os Casos de Testes a partir do PSM
	Validação	Executar os testes
		Avaliar os Resultados da Execução dos Testes

6. Trabalhos Relacionados

Dai (2004) apresenta um resultado da utilização do MDT. Mas, o artigo não deixa claro o que fazer para integrar o desenvolvimento e os modelos de teste.

Alves, Machado e Ramalho (2008), discutem objetivos e questões fundamentais sobre como integrar processos de MDD e MDT. Porém, não apresentam um processo ou metodologia de realização para a integração de ambas abordagens.

Por fim, em Asadi *et al.* (2010), apresenta-se uma proposta de processo de *software* utilizando MDA, apesar de citar algumas atividades de testes, o trabalho não apresenta um processo de teste de *software*.

7. Considerações Finais

Este artigo apresentou uma proposta de modelo de desenvolvimento de *software* dirigido por modelos. Durante o desenvolvimento desse trabalho foram identificados trabalhos que abordam processos de desenvolvimento de *software* e de testes de *software* dirigidos por modelos, de forma separada, não mostrando uma integração entre estes processos. Exceto o trabalho de Alves, Machado e Ramalho (2008), que apresenta parcialmente uma proposta de integração entre os processos. As primeiras etapas da pesquisa foram concluídas, as quais correspondem à revisão bibliográfica sobre o tema, as abordagens a serem utilizadas, bem como, a elaboração da proposta. Encontra-se em andamento as seguintes etapas: (i) detalhamento do modelo de processo de desenvolvimento de *software* proposto; (ii) definição do processo para realização dos estudos experimentais; e, (iii) o modelo *Qualitas* está em processo de validação por meio de estudo de caso no processo de desenvolvimento do Sistema de Triagem Neonatal do Hospital Universitário do Estado de Sergipe.

Os benefícios esperados pela utilização do modelo é permitir uma execução integrada do MDT com o MDD, bem como, um aumento no uso de modelos tanto no desenvolvimento quanto no teste de *software.* O *Qualitas* está sendo validado Como trabalhos futuros, deseja-se validar o modelo proposto em estudos e contextos reais.

Referências

ALVES, E. L. G.; MACHADO, P. D. L.; RAMALHO F. (2008). Uma Abordagem Integrada para Desenvolvimento e Teste Dirigido por Modelos. *In: 2nd Brazilian Workshop on Systematic and Automated Software Testing*.

AMELLER, D. (2009). *Considering Non-Functional Requirements in Model-Driven Engineering. Tesi de Màster. Universitat Politècnica de Catalunya*.

ÁVILA, A. L.; SPÍNOLA, R. O. (2007). Introdução à Engenharia de Requisitos. Engenharia de Software Magazine. Ano 1 - 1ª Edição. Pg. 46 a 51.

BAKER, P.; DAI, Z. R.; GRABOWSKI, J.; HAUGEN, Ø.; SCHIEFERDECKER, I.; WILLIAMS, C. (2008) *Model-Driven Testing Using the UML Testing Profile.* Springer-Verlag. New York.

BARTIÉ, A. (2002). Garantia da Qualidade de Software: adquirindo maturidade organizacional. Rio de Janeiro. Elsevier.

BASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. (2007) Base de Conhecimento em Teste de *Software*. 2ª Ed. ver. São Paulo: Martins.

BATH, G.; MCKAY, J. (2011) *The Software Test Engineer's Handbook: A Study Guide for the ISTQB Test Analyst and Technical Test Analyst Advanced Level Certificates*. ISBN 978-1-933952-24-6. USA.

BOEHM, B. W. (1988). A Spiral Model of Software Development and Enhancement. IEEE.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I., (1999). *The Unified Modeling Language User Guide, Addison-Wesley*.

BUARQUE, A. S. M. (2009). Desenvolvimento de *Software* Dirigido por Modelos: Um Foco em Engenharia de Requisitos. Universidade Federal de Pernambuco.

CRAG SYSTEMS. (s.d) *Model-Driven Development Process*. Disponível em: < http://www.cragsystems.co.uk/development_process/>. Acesso em 26 de Maio de 2013.

DATASUS. Processo de Teste de Software do DATASUS. (2008). Disponível em: < http://pts.datasus.gov.br/PTS/default.php?area=04>. Acesso em: 30/10/2013.

DÍEZ, A. B. G.; URICARTE, X. L. (2004). *Process Engineering and Project Management for the Model Driven Approach*. *In: 1st European Workshop*.

FONDEMENT F.; SILAGHI R. (2004). *Defining Model Driven Engineering Processes. In: 3rd Workshop in Software Model Engineering at the 7th International Conference on the UML*.

FOWLER, M. (2005). *Language Workbenches and Model Driven Architecture.*

FRANCE, R. B.; GHOSH, S.; DINH-TRONG, T.; SOLBERG, A. (2006). *Model-Driven Development Using UML 2.0: Promises and Pitfalls*. IEEE.

FRANCE, R.; RUMPE, B. (2007). Model-Driven Development of Complex Software: A Research Roadmap. In: 29th International Conference on Software Engineering - Future of Software Engineering. Minneapolis, USA: IEEE Computer Society.

GAVRAS, A.; BELAUNDE, M.; STEINHAU, R.; ALMEIDA, J. P. (2003). *MODA-TEL Model-Driven Methodology*, MODA-TEL IST-2001-37785.

GHOLAMI, M. F.; RAMSIN, R. (2010). *Strategies for Improving MDA-Based Development Processes. In: International Conference on Intelligent Systems, Modelling and Simulation*.

HECKEL, R.; LOHMANN, M. (2003). *Towards Model-Driven Testing. Electronic Notes in Theoretical Computer Science*, 82 N° 6. Elsevier Science.
HILDENBRAND, T.; KORTHAUS, A. (2004). *A Model-Driven Approach to Business Software Engineering.*

HIRAMA, K. (2012). Engenharia de Software: Qualidade e Produtividade com Tecnologia. Rio de Janeiro: ELSEVIER.

IBM (2003).*Overview Model-driven Testing Tools. Release 4.1*.

IEEE/IEC 12207 (2008). *Systems and Software Engineering — Software Life Cycle Processes. Second Edition*.

ISTQB. *Certified Tester: Advanced Level Syllabus. Version* 2007. Disponível em: < http://www.istqb.org/downloads/viewcategory/23.html>. Acesso em 10/01/2014.

_____. *Certified Tester: Foundation Level Syllabus. Version* 2010. Disponível em: <http://www.istqb.org/downloads/finish/16/15.html>. Acesso em 15/12/2013.

_____. *Certified Tester Advanced Level Syllabus Test Analyst*. *Version* 2012a. Disponível em: < http://www.istqb.org/downloads/viewcategory/46.html>. Acesso em 22/11/2013.

_____. *Certified Tester Advanced Level Syllabus Test Manager*. *Version* 2012b. Disponível em: < http://www.istqb.org/downloads/viewcategory/46.html>. Acesso em 27/11/2013.

JAVED A. Z.; STROOPER, P. A.; WATSON, G. N. (2007). *Automated Generation of Test Cases Using Model-Driven Architecture.* KENT, S. (2003). *Model Driven Engineering. Integrated Formal Methods. In: Third International Conference*. LNCS Vol. 2335, Springer-Verlag.

KLEPPE, A.; WARMER J.; BAST, W. (2003). *MDA Explained: The Practice and Promise Of The Model Driven Architecture. Addison Wesley*,

KUHN, T.; GOTZHEIN, R.; WEBEL, C. (2006). *Model-Driven Development with SDL – Process, Tools, and Experiences, in Model Driven Engineering Languages and Systems. Springer LNCS.*

LARRUCEA, X.; DÍEZ, A. B. G.; MANSELL, J. X. (2004). *Practical Model Driven Development Process*.

LIMA, H. S.; RAMALHO, F.; MACHADO, P. D. L.; GALDINO, E. L. (2007). *Automatic Generation of Platform Independent Built-in Contract Testers. Formal Methods Group*.

MACIEL, C. L.; MACHADO, P. D. L.; RAMALHO, F. (2009). Uma Técnica MDT para a Geração Automática de Casos de Teste Usando Padrões de Teste. *In:* Workshop Brasileiro de Teste de *Software* Sistemático e Automatizado.

MACIEL, C. L. (2010). Uma Abordagem Dirigida por Modelos para a Geração Automática de Casos de Teste de Integração Usando Padrões de Teste. Universidade Federal de Campina Grande.

MAGALHÃES, A. P.; DAVID, J. M. N.; MACIEL, R. S. P.; SILVA, F. A. (2011) *Modden: An Integrated Approach for Model Driven Development and Software Product Line Processes*.

MELLOR, S. J.; SCOOT, K.; UHL, A.; WEISE, D. (2005). MDA Destilada: Princípios de Arquitetura Orientada por Modelos. Rio de Janeiro: Editora Ciência Moderna Ltda.

MILICEV, D. (2009). *Model-Driven Development with Executable UML.*

MOHAN, P.; SHANKAR, A. U.; JAYASRIDEVI, K. (2012). Quality Flaws: Issues and Challenges in Software Development. Computer Engineering and Intelligent System. ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online). Vol 3, Nº 12.

MOLINARI, L. (2008) Testes Funcionais de *Software.* Editora: *Visual Books*. Florianópolis.

MYERS, G. (2004) *The Art of Software Testing*. *Revised and updated* by Tom Badgett and Todd Thomas, with Corey Sandler. 2nd ed. ISBN 0-471-46912-2. Addison-Wesley.

NIKULSINS, V.; NIKIFOROVA, O. (2008). *Adapting Software Development Process Towards the Model Driven Architecture. In: The Third International Conference on Software Engineering Advances*.

OMG. *MDA Guide Version* 1.0.1., 2003.

_____. *OMG Model Driven Architecture*. (2013). Disponível em: <http://www.omg.org/mda/>. Acesso em 19/11/2013.

PÁDUA, W. (2003) Engenharia de Software: Fundamentos, Métodos e Padrões. LTC, 2ª edição.

PEZZÈ, M.; YOUNG, M. (2008). Teste e Análise de *Software:* Processos, Princípios e Técnicas. Editora Artmed. Porto Alegre.

PRESSMAN, R. S. (2010). Engenharia de *Software*. 6ª ed. – Porto Alegre.

RIOS, E.; MOREIRA, T. R. (2003). Teste de *Software*. Editora Alta Books. Rio de Janeiro.

SANTOS, P. A. NETO. (2006). Modest: Um Método de Teste Baseado em Modelos. Universidade Federal de Minas Gerais.

SCHACH, S. R. (2008). Engenharia de Software: Os Paradigmas Clássico e Orientado a Objetos. 7ª Edição. Editora Mc Graw Hill. São Paulo.

SELIC, B. (2003). *The Pragmatics of Model-Driven Development.*

SOMMERVILLE, I. (2007) Engenharia de *Software.* 8ª ed. – São Paulo: Pearson Addison-Wesley.

SOUSA, H. C. S. (2009). Construção Automatizada de Casos de Teste usando Engenharia Dirigida por Modelo, Dissertação de Mestrado em Engenharia de Eletricidade, área de concentração Ciência da Computação). Universidade Federal do Maranhão.

SOUSA, H. C. S.; LOPES, D.; ABDELOUAHAB, Z.; BARREIRO, D.; HAMMOUD S. (2011). *An Approach for Model Driven Testing: Framework, Metamodels and Tools*. In: *International Journal of Computer Systems Science & Engineering*.

SPILLNER, A.; LINZ, T.; SCHAEFER, H. (2011) *Software Testing Foundations: A Study Guide for the Certified Tester Exam.* ISBN 978-1-933952-78-9. Rocky Nook.

SWEBOK, *Guide to the Software Engineering Body of Knowledge. Version 2004.* Disponível em http://www.nt.fh-koeln.de/fachgebiete/inf/nissen/softeng/swebok.pdf. Acesso em 10/11/2013.

VASCONCELOS, A. M. L.; ROUILLER, A. C.; MACHADO, C. Â. F. e MEDEIROS, T. M. M. (2006). Introdução à Engenharia de *Software* e à Qualidade de *Software*. – Lavras: UFLA/FAEPE.

WAZLAWICK, Raul Sidnei. (2008). Metodologia de Pesquisa para Ciência da Computação. Rio de Janeiro, RJ, Editora: Elsevier, 2008. ISBN 978-85-352-3522-7.