Ministério da
Ciência e Tecnologia

GESTÃO DE CONHECIMENTO APLICADO A TESTE DE SOFTWARE: UMA ABORDAGEM BASEADA EM ONTOLOGIA

Érica Ferreira de Souza

Exame de Proposta de Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Nandamudi Lankalapalli Vijaykumar

Registro do documento original:
<http://urlib.net/xxx>

INPE
São José dos Campos
2011

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência e Tecnologia

GESTÃO DE CONHECIMENTO APLICADO A TESTE DE SOFTWARE: UMA ABORDAGEM BASEADA EM ONTOLOGIA

Érica Ferreira de Souza

Exame de Proposta de Tese de Doutorado do Curso de Pós-Graduação em Computação Aplicada, orientada pelo Dr. Nandamudi Lankalapalli Vijaykumar

Registro do documento original:
<http://urlib.net/xxx>

INPE
São José dos Campos
2011

AGRADECIMENTOS

À CAPES pelo apoio financeiro de julho de 2010 até agosto de 2011. E também à FAPESP por ter aprovado financiar esse trabalho de pesquisa em nível de doutorado a partir de setembro de 2011.

RESUMO

Organizações de desenvolvimento de software vêm buscando, cada vez mais, agregar qualidade aos produtos gerados. Os processos de teste são elementos estratégicos para a condução de projetos de desenvolvimento e qualidade do produto. Diante disso, tais organizações têm mostrado um crescente interesse por programas de melhoria de processos. Diversos métodos e modelos de melhoria de processos têm sido desenvolvidos para aumentar os níveis de maturidade organizacional do projeto. Porém, não basta somente atualizar o processo organizacional do projeto, é necessário que as lições aprendidas sejam processadas para serem utilizadas no apoio à tomada de decisão. É nesse contexto que a gestão do conhecimento tem apresentado um papel fundamental na melhoria de processos de software. No entanto, em sua grande maioria, trabalhos de gestão de conhecimento exploram o ambiente de desenvolvimento de software, e não especificamente o ambiente de teste de software. Diante disso, este trabalho, referente a proposta da tese de doutoramento, tem como objetivo investigar estratégias que possam promover uma melhoria de processos de teste de software. Para isso, propõe-se estruturar uma base de conhecimento que apoie o armazenamento, compartilhamento e reuso do conhecimento em ambientes de teste de software.

Palavras-chave: Teste de software, Processo de teste, Gestão de conhecimento e Melhoria de processos.

REUSE STRATEGIES FOR IMPROVING SOFTWARE TESTING PROCESSES BASED ON ONTOLOGIES

ABSTRACT

Software development organizations are seeking to add quality to their products. Testing processes are strategic elements to manage projects and product quality. Therefore, such organizations have shown an increasing interest in process improvement programs. Several methods and process improvement models have been developed to increase the levels of organizational maturity of the project. But it is just not enough to update the organizational process of the project; it is necessary that the lessons learned are processed for use in supporting decision making. In this context, knowledge management has had a major role in improving software processes. However, for most of the published literature deals with software development environments and processes, but not specifically with software testing environment and processes. Therefore, this thesis proposal investigates strategies to promote improving software testing process. The proposal investigates to organize and structure a knowledge base that enables storage, sharing and reuse of knowledge in software testing environments.

Keywords: Software Testing, Test Process, Knowledge management and Process Improvement.

LISTA DE ABREVIATURAS E SIGLAS

AMP Avaliação e Melhoria de Processos de Software

CMM *Capability Maturity Model*

CMMI *Capability Maturity Model Integration*

COMAER Comando da Aeronáutica

DAML *DARPA Agent Markup Language*

DCTA Departamento de Ciência e Tecnologia Aeroespacial

GC Gestão de Conhecimento

GQM *Goal Questions Measures*

GQ(I)M *Goals Questions (Indicators) Measures*

ICA-MMH Integração e Cooperação Amazônica para Modernização do Monitoramento Hidrológico

IDEAL *Initiating, Diagnosing, Establishing, Acting and Learning*

IEEE *Institute of Electrical and Electronics Engineers*

INPE Instituto Nacional de Pesquisas Espaciais

KBSE *Knowledge-Based Software Engineering Conference*

MPS.BR Melhoria de Processo do Software Brasileiro

MPT.BR Melhoria de Processo de Teste Brasileiro

ODE *Ontology-based software Development Environment*

OIL *Ontology Interchange Language*

OPTS Ontologia de Processo de Teste de Software

OTM3 *Organizational Testing Management Maturity Model*

OWL *Ontology Web Language*

PDCA *Plan/Do/Check/Action*

RDF *Resource Description Framework*

SABiO *Systematic Approach for Building Ontologies*

SEI *Software Engineering Institute*

SPARQL *Protocol and RDF Query Language*

SQWRL *Semantic Query-enhanced Web Rule Language*

TI Tecnologia de Informação

TIM *Test Improvement Model*

TMM *Test Maturity Model*

TPI *Test Process Improvement*

UML *Unified Modeling Language*

VLS Veículo Lançador de Satélites

XML *eXtensible Markup Language*

W3C *Word Wide Web Consortium*

SUMÁRIO

Pág.

1 INTRODUÇÃO

Durante as últimas décadas, devido ao surgimento de novas tecnologias em diversas aplicações, técnicas mais avançadas vem sendo empregadas no desenvolvimento de software. Em função disso, processos para desenvolver e qualificar o software, devem ser embutidos. Isto aplica-se a sistemas críticos, como espaciais, sistemas de controle em usinas nucleares, equipamentos médicos, entre outros. Desta forma, é fundamental que um processo, claramente definido, seja aplicado ao ambiente de desenvolvimento do software. Um processo de desenvolvimento de software é um conjunto de etapas necessárias para transformar requisitos em software (PRESSMAN, 2006). Já a garantia de qualidade do software depende da forma como são conduzidas essas etapas, principalmente a etapa inerente às **atividades de teste**.

A falta do uso de mecanismos que garantam a qualidade dos software pode causar perdas significativas. Alguns exemplos de falhas em softwares considerados críticos são: a série de processadores *Pentium Intel*, devido a um defeito no co-processador matemático (Unidade de Ponto Flutuante), ficando conhecido como "*Pentium FDIV bug*" ou "Bug de ponto flutuante"; e a autodestruição do foguete "Ariane 5", devido a um erro no software de controle (PECHEUR, 2000).

Diversas organizações estão despendendo de esforços significativos na avaliação e melhoria de seus processos de desenvolvimento de software, devido ao fato de estudos mostrarem que a garantia de qualidade do processo de desenvolvimento influencia significativamente na qualidade do software gerado (FUGGETTA, 2000). Bons processos ajudam a produzir software de melhor qualidade, com maior agilidade e com custos menores (KOOMEN; POL, 1999). Visando melhorar os processos de desenvolvimento de software aplicados na organização, métodos e modelos de melhoria do processo têm sido desenvolvidos. Dentre eles, destacam-se: *Initiating, Diagnosing, Establishing, Acting and Learning* (IDEAL), (GREMBA; MYERS, 1997), *Capability Maturity Model Integration* (CMMI), (SEI, 2006) e Melhoria de Processo do Software Brasileiro (MPS.BR), (SOFTEX, 2009).

Considerando a evolução tecnológica e a busca de maturidade da organização em seus processos de engenharia de software, os sistemas considerados grandes e complexos não só necessitam de melhoria nos padrões de processo referente ao desenvolvimento de software, mas também em relação aos padrões de **processos de teste**. A atividade de teste de software, devido a sua importância, também tem desper-

tado interesse de pesquisadores e profissionais da área. Assim como o processo de desenvolvimento de software, a atividade de teste também está incorporada a um processo, pois é composta por diversas etapas com o objetivo de agregar qualidade aos produtos gerados (BASTOS et al., 2007). Sendo a atividade de teste um processo, a melhoria deste também deve ser considerada.

Alguns modelos de melhoria de processo de teste, também chamados **modelos de maturidade**, são referências nas organizações para a avaliação de processos de teste. Alguns exemplos são: *Test Maturity Model* (TMM), (BURNSTEIN et al., 1996), *Test Improvement Model* (TIM), (ERICSON et al., 1997), *Test Process Improvement* (TPI), (ANDERSIN, 2004) e Melhoria de Processo de Teste Brasileiro (MPT.BR), (MPT.BR, 2009). Segundo SEI (2006), os modelos de maturidade sugerem medições para a avaliação dos processos, sendo o nível de maturidade relacionado com a sua estrutura e controle organizacional.

Atualmente, realizar a melhoria de processos baseando-se apenas nos níveis de maturidade organizacional gerados pelos modelos de melhoria, não é suficiente. Não basta somente atualizar o processo organizacional do projeto. É necessário que as lições aprendidas durante esses processos sejam disseminadas por toda a organização (BORGES; FALBO, 2002). A comunidade acadêmica deseja explorar dados[1] coletados ao longo dos projetos. Tais dados podem fornecer informações importantes para a tomada de decisão, envolvendo a identificação e a realização de ações corretivas que podem promover melhoria dos processos. Estas informações, que auxiliam na tomada de decisão, podem ser incorporadas em alguma base de conhecimento para facilitar a sua gestão e seu acesso.

É nesse contexto que a **Gestão de Conhecimento (GC)** apresenta um papel fundamental na melhoria de processos de desenvolvimento de software. Segundo Natali e Falbo (2002), a GC pode apoiar as atividades de planejamento e controle da qualidade, pois facilita a criação, acesso e reuso do conhecimento. O conhecimento tratado de forma eficiente é capaz de apoiar as tomadas de decisões (MARKKULA, 1999). Esta abordagem vem sendo investigada por diversos trabalhos recentes (SANTOS et al., 2007; MORO; FALBO, 2009; ANDRADE et al., 2010). No entanto, em sua grande maioria, trabalhos de gestão de conhecimento exploram o ambiente de de-

[1] Alguns dados que podem ser coletados são: tempo gasto para um processo, em particular, ser concluído; recursos requeridos, como esforço total por pessoa e gastos com equipamentos; ferramentas; e o número de ocorrências decorrente de um evento como, mudanças nos requisitos.

senvolvimento de software e não, especificamente, o ambiente de teste.

1.1 Motivação

A área de testes é destaque em ambientes acadêmicos e industriais. No âmbito espacial, as pesquisas nesta área também vem sendo intensificadas (AMBRóSIO et al., 2008). O tema sugerido neste trabalho de pesquisa possui considerada relevância nos projetos espaciais do Instituto Nacional de Pesquisas Espaciais (INPE), envolvendo softwares críticos embarcados nos equipamentos científicos a bordo de satélites e/ou balões estratosféricos. Os resultados desta proposta de trabalho de pesquisa, também poderão beneficiar as pesquisas do Departamento de Ciência e Tecnologia Aeroespacial (DCTA), órgão subordinado ao Comando da Aeronáutica (COMAER), incumbido, dentre outros, pela fabricação de Veículo Lançador de Satélites (VLS).

No INPE e no DCTA, os setores ligados à construção de satélites e VLS contam com algum tipo de arquitetura ou até mesmo ambientes automatizados para qualificar os softwares considerados críticos (SANTIAGO et al., 2008; LAMAS, 2010). Porém, devido ao grande volume de dados gerados por esses ambientes, nem sempre é possível encontrar correlação, de uma forma manual, entre as informações, gerando conhecimento adequado para determinar ações de melhoria na reutilização do processo de teste. Neste sentido, destaca-se a importância de priorizar pesquisas que contribuam com a área de processos de teste de software, garantindo assim uma maior qualidade do produto de software gerado.

1.2 Objetivo do trabalho

Dentro deste contexto, o trabalho tem como objetivo geral investigar estratégias que visam promover uma melhoria de processos de teste de software a partir de princípios de GC. Para que a aplicação da estratégia seja efetiva propõe-se uma base de conhecimento (repositório de experiências) que apoie o armazenamento, o compartilhamento e o reuso do conhecimento. A base de conhecimento armazenará informações pertinentes sobre o processo de teste corrente da organização.

Para o cumprimento deste objetivo geral, propõe-se as seguintes metas específicas:

- Caracterizar o processo de teste, identificando características comuns do domínio;

- Criar uma ontologia que propicie a organização e a estruturação do conhecimento gerado a partir da caracterização do processo de teste;

- Criar instâncias da ontologia;

- Estruturar uma base de conhecimento, a partir da ontologia, que agregue as diversas informações identificadas no processo de teste; e

- Analisar os benefícios e as oportunidades de melhoria para a estratégia proposta em estudos de caso na indústria e também no âmbito espacial.

Uma vez alcançados os objetivos acima, podem ser realizados diversos experimentos com a estrutura de conhecimento criada, promovendo um entendimento fácil aos especialistas do domínio sobre o processo, podendo tirar melhor proveito das interpretações conceituais da ontologia criada. A base de conhecimento possibilitará minerações, visualizações informativas, pesquisas e análises para serem utilizadas como melhoria dos processos em projetos futuros.

1.3 Organização do Texto

A divisão dos capítulos deste trabalho está descrita a seguir:

- CAPÍTULO 2 - GESTÃO DE CONHECIMENTO NA QUALIDADE DO SOFTWARE: descreve brevemente GC e ontologia. Nesse capítulo também são apresentados trabalhos que lidam com a GC aplicada na melhoria de processos de desenvolvimento de software.

- CAPÍTULO 3 - TESTE DE SOFTWARE: são mostrados de forma sucinta os principais conceitos relacionados às atividades de teste de software. Este capítulo procura, também, dar uma visão geral de métodos e modelos de melhoria de processos de teste. Por fim, são apresentados alguns trabalhos que envolvem o uso da GC no ambiente de teste de software.

- CAPÍTULO 4 - ESTRATÉGIA PARA POSSIBILITAR A MELHORIA DE PROCESSO DE TESTE: é apresentado a metodologia desta proposta de trabalho de pesquisa. O capítulo é dedicado também a colocar as contribuições aqui apresentadas na perspectiva de uma tese de doutorado.

- CAPÍTULO 5 - CRONOGRAMA DO TRABALHO: são apresentadas as principais atividades realizadas e previstas para serem desempenhadas até a conclusão do trabalho de pesquisa em nível de doutorado e o cronograma de atividades.

2 GESTÃO DE CONHECIMENTO NA QUALIDADE DO SOFTWARE

Com o crescimento exponencial de dados vindos de diversas fontes de conhecimento distintas dentro de uma organização, torna-se necessário a automatização das tarefas de aquisição, processamento, análise e disseminação do conhecimento. As organizações necessitam gerenciar de forma efetiva as informações geradas em seu ambiente de produção para promover a melhoria dos processos utilizados e gerar conhecimento que de suporte às decisões futuras.

Diante deste contexto, a GC exerce um papel fundamental para as organizações. A GC tem a capacidade de reter e gerir conhecimento, tornando-se um dos fatores mais importantes no desenvolvimento de soluções. O principal objetivo da GC é promover o surgimento de conhecimento novo, seu armazenamento e compartilhamento por toda a organização (O'LEARY; STUDER, 2001). Na área de engenharia de software não é diferente. Atualmente, diversas organizações de desenvolvimento de software buscam utilizar o conhecimento para desenvolverem competências específicas e capacidade inovadora, que se traduzem em serviços com menor custo e tempo, e produtos com maior índice de qualidade.

Uma das características de projetos de engenharia de software é o elevado número de informações que são geradas e manipuladas. Os envolvidos no projeto enfrentam problemas, tais como: dificuldade de sistematizar as informações geradas ao longo dos processos de software; dificuldade para reutilizar o conhecimento gerado de um projeto em outro, por falta de vocabulário comum; perda de capital intelectual da organização, devido à rotatividade das equipes; e a não representação do conhecimento (ANDRADE et al., 2010). A expectativa é que o emprego da GC amenize ou resolva, pelo menos parcialmente, estes problemas.

Muitas pesquisas relacionadas à melhoria de processos tem sido realizadas pela comunidade de engenharia de software. Várias delas, inclusive, defendem a GC como fator fundamental para o aumento da qualidade e produtividade. Esta preocupação é uma das principais razões para o grande crescimento da pesquisa em GC. Nesta seção são apresentadas as principais tendências do uso da GC para a melhoria de processos de software. Inicialmente, é apresentada uma visão geral dos principais conceitos relacionados à GC e posteriormente são apresentados trabalhos correlatos que exploram a aplicação de GC na melhoria de processos de software.

2.1 Gestão de Conhecimento (GC)

O conhecimento é um fator que sempre esteve presente na história da humanidade. O termo tornou-se mais presente dentro das organizações na década de 80, vindo da necessidade de obter conhecimento a partir da grande quantidade de informação para desenvolver novos produtos, processos e arranjos organizacionais mais flexíveis, proporcionando uma vantagem competitiva sustentável (SPENDER, 1996). Ter controle e facilidade no acesso ao conhecimento, tornou-se hoje, um diferencial competitivo e econômico para as organizações.

De acordo com Markkula (1999), conhecimento é informação combinada com experiência, interpretação e reflexão. É uma forma de informação pronta para ser aplicada na tomada de decisões. Existem também definições para "conhecimento organizacional", que é a informação processada e embutida em rotinas e processos que possibilitam ações. É também o conhecimento capturado pelos sistemas, processos, produtos, regras e cultura da organização (MYERS, 1996).

Dentre os principais conceitos que envolvem GC é necessário compreender a diferença entre três deles: dado, informação e conhecimento. Cada um desses elementos funciona como base para a existência do elemento seguinte. Há vários autores que buscam destacar a diferença existente entre esses conceitos, porém não existe propriamente um consenso quanto à diferenciação ou definição.

Segundo Davenport e Prusak (1998), os dados representam fatos distintos e objetivos, relativos a eventos ocorridos em uma organização, é também conteúdo bruto de uma informação. As informações são dados dotados de relevância e propósito, exercendo alguma influência sobre o julgamento do indivíduo que as utiliza. Por sua vez, a informação torna-se conhecimento, podendo ser usada para fazer previsões, ou seja, o conhecimento tem embutido em si valores como sabedoria. Desse modo, Davenport e Prusak (1998) propõe a seguinte diferenciação de dado, informação e conhecimento, apresentada na Tabela 2.1:

Tabela 2.1 - Diferenciação entre dado, informação e conhecimento (DAVENPORT; PRUSAK, 1998)

DADO	INFORMAÇÃO	CONHECIMENTO
Simples observação sobre o estado do mundo	Dados dotados de relevância e propósito	Informação valiosa da mente humana. Inclue reflexão, síntese e contexto
Facilmente estruturado	Requer unidade de análise	Difícil estruturação
Facilmente obtido por máquinas	Exige consenso em relação ao significado	Difícil captura em máquinas
Frequentemente quantificado	Exige necessariamente a medição humana	Frequentemente tácito
Facilmente transferível		Difícil transferência

O conhecimento pode ser classificado como tácito e explícito (NONAKA; TAKEUCHI, 1997). O conhecimento tácito surge de experiências individuais e envolve fatores como crença pessoal, perspectivas e valores. Envolve fatores intangíveis, por exemplo, intuições e emoções. O conhecimento explícito, por sua vez, pode ser expresso de uma forma estruturada, tais como tabelas, figuras, desenhos, esquemas, diagramas e requisitos. As formas como ocorrem a interação entre o conhecimento tácito e explícito e entre a organização e o indivíduo, podem ser expressas em quatro processos de conversão de conhecimento que constituem uma espiral do conhecimento, proposta por Nonaka e Takeuchi (1997). Os quatro passos são:

- Socialização: troca de experiência, ou seja, conversão de parte do conhecimento tácito de uma pessoa no conhecimento tácito de outra pessoa. O aprendizado se dá por meio da observação, imitação e prática. Segue um modelo mestre-aprendiz;
- Externalização: conversão de parte do conhecimento tácito do indivíduo em algum tipo de conhecimento explícito. A externalização é considerada a chave para a criação do conhecimento, cria conceitos novos e explícitos a partir do conhecimento tácito e facilita a transmissão dos conhecimentos tácitos que são, geralmente, de difícil comunicação;
- Combinação: conversão de algum tipo de conhecimento explícito gerado por um indivíduo para agregá-lo ao conhecimento explícito da organiza-

ção. Esse modo de conversão do conhecimento envolve a combinação de conjuntos diferentes de conhecimento explícito, por exemplo, a classificação, sumarização, pesquisa e categorização das informações com a utilização da tecnologia de banco de dados e pode levar à criação de novos conhecimentos; e

- Internalização: conversão de partes do conhecimento explícito da organização em conhecimento tácito do indivíduo. Aos conhecimentos explícitos são adicionados novos conhecimentos tácitos por meio de diversos tipos de experiências.

A espiral, formada quando os ciclos de conversão de conhecimento passam várias vezes por esses quatro módulos, serve para analisar e entender os mais diversos casos de criação e disseminação do conhecimento, sendo que cada caso terá suas particularidades ou especificidades. A Figura 2.1, apresenta o espiral do conhecimento e o conteúdo criado pelas quatro formas de conversão.

Figura 2.1 - Espiral do conhecimento e o conteúdo criado pelas quatro formas de conversão do conhecimento.
Fonte: Adaptado de Nonaka e Takeuchi (1997)

A abordagem teórica da criação do conhecimento é uma contribuição relevante e ponto de partida para trabalhos de diversas áreas. No contexto da engenharia de software são estudadas áreas, como: *i*) gestão de competências; *ii*) capacitação e aprendizado da equipe; *iii*) ambiente de colaboração; *iv*) portais; *v*) formas de distribuição do conhecimento; e *vi*) reestruturação e **melhoria de processos**.

Em um ambiente de engenharia de software uma quantidade enorme de informação é produzida ao longo do processo de desenvolvimento de um software. Estas informações precisam ser compartilhadas para auxiliarem em processos de aprendizagem, para tornar mais ágil a resolução de problemas e para desenvolver novos produtos. No entanto, transformar toda esta informação em conhecimento aplicável não é uma tarefa fácil, por se tratar de um grande volume. Surge então a necessidade de **representar o conhecimento**, para torná-lo acessível e tratável.

2.2 Representação do Conhecimento

Segundo Davis et al. (1993), a definição de representação do conhecimento pode ser compreendida em termos de cinco papéis distintos que esta representa:

1. Um representação de conhecimento é um mecanismo usado para raciocinar sobre o mundo ao invés de agir diretamente sobre ele.
2. É uma resposta à pergunta "Em que termos devo pensar sobre o mundo?".
3. É uma teoria fragmentária do raciocínio, expresso em termos de três componentes: *i*) a representação é fundamental concepção de inferência inteligente; *ii*) o conjunto de inferências que a representação sanciona; e *iii*) o conjunto de inferências que ela recomenda.
4. É um meio de computação eficiente.
5. É um meio de expressão humana, isto é, uma linguagem em que dizemos coisas sobre o mundo.

O maior desafio na representação do conhecimento é que boa parte do conhecimento em engenharia de software é tácito (não documentado) e, dificilmente, são dedicados esforços para torná-lo explícito. Quando o conhecimento tácito é transformado em conhecimento explícito, a organização começa a viabilizar a disseminação de um conhecimento estratégico.

A inexistência de padronização na representação do conhecimento dificulta a sua compreensão pelos diversos atores envolvidos. Para apoiar as atividades envolvidas na GC, diversas tecnologias podem ser aplicadas, como Banco de Dados, Máquinas de Busca, *Internet* e *Intranet*, entre outras (O'LEARY, 1998a). Porém, uma tecnologia particularmente importante para a representação do conhecimento, muito estudada nos últimos anos, é a **ontologia**.

Segundo Rios (2005), o estabelecimento de ontologias pode contribuir para a estruturação e classificação das informações. Construindo-se ontologias, é possível estabelecer significados formais para alguns termos do vocabulário, isto facilita a reutilização do conhecimento em contextos não previstos antecipadamente (FALBO, 1998).

2.2.1 Ontologia

A literatura apresenta uma série de definições distintas para ontologia. De acordo com Gruber (1993), a ontologia é uma especificação formal de uma conceituação compartilhada. Já Guarino (1998), acrescenta que o grau de especificação de uma conceituação de uma linguagem utilizada para uma base de conhecimento depende do propósito desejado para a ontologia.

Ontologias envolvem a descrição de conceitos em um determinado domínio de conhecimento, com suas propriedades e restrições. A ontologia fornece uma descrição exata do conhecimento em uma linguagem formal para facilitar a comunicação, integração, busca, armazenamento, compartilhamento, reutilização e representação do conhecimento (O'LEARY, 1998b). O grande interesse em ontologias, hoje, dá-se pela necessidade de haver, cada vez mais, uma maior interoperabilidade e reutilização de informações entre os sistemas e as pessoas dentro de uma organização (RIOS, 2005).

Classificação

São várias as classificações de ontologia encontradas na literatura. Guarino (1998) define a seguinte classificação:

- Ontologias Genéricas: Também chamada ontologia de fundamentação ou ontologia de nível superior, é uma conceituação abstrata sobre elementos genéricos para fazer parte de um domínio específico, como espaço, tempo, matéria, objeto, evento e ação.

- Ontologias de Domínio: Descrevem o vocabulário relacionado a conceituações de domínios particulares, como medicina ou direito.

- Ontologias de Tarefa: Similar a uma ontologia de domínio, porém, ao invés de mapear os conceitos de um domínio particular, mapeia conceitos de uma tarefa ou atividade específicas, como diagnóstico, venda e matrícula.

- Ontologias de Aplicação: Descrevem conceitos que são dependentes de um domínio e de uma tarefa particular e, assim, combina especializações de conceitos presentes nas ontologias de domínio e de tarefa.

A Figura 2.2 apresenta os quatro tipos de ontologias e seus relacionamentos. No topo estão as ontologias genéricas que definem conceitos gerais que embasam todas as outras conceituações e no nível mais baixo encontram-se as ontologias de aplicação, que especializam conceituações para uma determinada classe de aplicações.

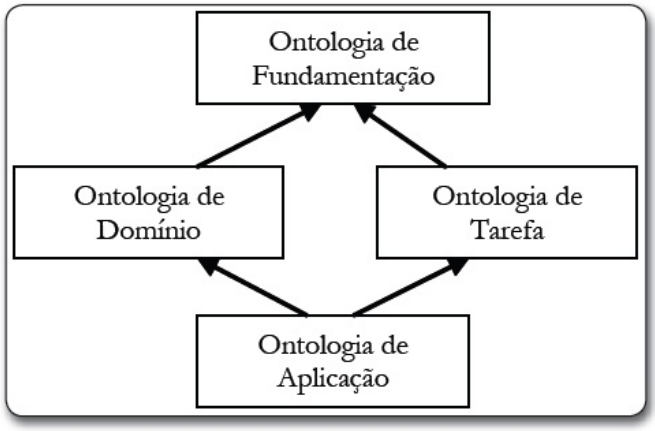

Figura 2.2 - Relacionamentos entre os tipos de ontologias.
Fonte: Adaptado de Guarino (1998)

Ferramentas

Existem diversas ferramentas de apoio à construção de ontologias, por exemplo, a Protégé 2000 (NOY; MUSEN, 2000), a *OntoEdit* (STAAB; MAEDCHE, 2000) e a *Text-to-onto* (MAEDCHE; VOLZ, 2001). Em geral, as ferramentas utilizam linguagem de representação para a construção das ontologias.

Linguagens de Representação

Para representar uma conceitualização compartilhada, é necessária uma linguagem de representação. Existem hoje muitas linguagens de representação definidas e a grande maioria é baseada na sintaxe do *eXtensible Markup Language* (XML) (W3C, 1996), que tende a tornar-se a linguagem padrão para modelar as ontologias. Algumas das linguagens existentes estão detalhadas na Tabela 2.2.

Tabela 2.2 - Linguagens de representação de Ontologias

Linguagem	**Descrição**
RDF/RDF *Schema*	A *Resource Description Framework* (RDF)/ RDF *Schema* foi desenvolvida pela *Word Wide Web Consortium* (W3C) (RDF, 2004). Foi criada com o objetivo de representar o conhecimento utilizando redes semânticas. São linguagens que permitem a representação de conceitos, taxonomias de conceitos e relações binárias. A primeira define como descrever recursos através de suas propriedades e valores, enquanto a segunda é responsável por fornecer mecanismos para declarar as propriedades RDF. Pode-se entender o *Schema* como sendo um dicionário que define os termos a serem utilizados em uma declaração RDF. Baseiam-se na notação em XML.
OIL	A linguagem *Ontology Interchange Language* (OIL) foi proposta pelo projeto *On-to-Knowledge* (FENSEL; HARMELEN, 2001). Combina primitivas de modelagem das linguagens baseadas em *frames* com a semântica formal e serviços de inferência da lógica descritiva. Pode verificar classificação e taxonomia de conceitos.
DAML + OIL	Sucessor da linguagem OIL, a linguagem *DARPA Agent Markup Language* (DAML) + OIL foi desenvolvida como uma extensão para linguagem RDF, acrescentando construtores mais expressivos. Provê uma infraestrutura básica que permite às máquinas fazerem a mesma classificação de inferências simples que os seres humanos fazem (HORROCKS, 2002).

continua na próxima página

Tabela 2.2 - Linguagens de representação de Ontologias

Linguagem	Descrição
OWL	A *Ontology Web Language* (OWL) é uma linguagem de marcação semântica utilizada para representar explicitamente o conjunto de termos de um vocabulário e os relacionamentos que existem entre eles (OWL, 2003). É especialmente indicada quando há necessidade de descrever ontologias e para ser usada em aplicações Web que necessitam processar o conteúdo de informações disponíveis. A OWL é desenvolvida como uma extensão do vocabulário da RDF e é derivado do DAML + OIL.

Métodos de construção de ontologias

Os métodos de construção são desenvolvidos no intuito de sistematizar a construção e a manipulação de ontologias. Diversos métodos para construção de uma ontologia foram propostos, por exemplo, *Sensus* (SWARTOUT et al., 1996) e *On-to-knoweledge* (FENSEL et al., 2000). Com a finalidade de reunir em uma única proposta, as melhores características dos métodos existentes, Falbo (2004a) apresentou um método para a construção de ontologias, chamado método *Systematic Approach for Building Ontologies* (SABiO). As principais fases do método são:

- Propósito e Especificação de Requisitos: identificar a competência da ontologia e analisar aquilo que é relevante. Inclui a definição de questões de competência que indicam as questões que a ontologia deve ser capaz de responder;
- Captura da Ontologia: consiste em capturar a conceituação do domínio, com base na competência da ontologia. Conceitos, relações, propriedades e axiomas[1] relevantes devem ser identificados e organizados. Modelos usando uma linguagem gráfica e um dicionário de termos devem ser utilizados para facilitar a comunicação com especialistas do domínio;
- Formalização da Ontologia: representar a conceituação capturada pela ontologia em uma linguagem formal;

[1]Os axiomas especificam definições de termos na ontologia e restrições sobre sua interpretação, bem como as leis de integridade que os regem (FALBO, 2004a)

- Integração de Ontologias Existentes: reutilizar conceituações de outras já existentes, de modo a reutilizar conceituações previamente estabelecidas, quando necessário;

- Avaliação da Ontologia: uma ontologia deve ser avaliada, com o propósito de verificar se os requisitos estabelecidos na especificação são satisfeitos; e

- Documentação da Ontologia: o desenvolvimento da ontologia deve ser documentado com propósitos, requisitos, descrições textuais da conceituação, ontologia formal e critérios de projeto.

O método SABiO inclui, ainda, o uso de uma linguagem de modelagem para facilitar a comunicação dos modelos da ontologia, sugerindo um perfil *Unified Modeling Language* (UML) como linguagem gráfica para representação das ontologias.

Áreas aplicadas

A ontologia é aplicada em diversas áreas e muitos projetos são desenvolvidos, com a intenção de criar ontologias genéricas que possam ser utilizadas por diversos sistemas em todo o mundo. Todas as ontologias criadas tem sempre o mesmo foco, criar facilidades para aquisição, manutenção e acesso a informações vindas de diversas fontes, promovendo a sua reutilização. Algumas áreas de aplicação da ontologia conhecidas são: *i*) Processamento de linguagem natural; *ii*) Web Semântica; *iii*) Comércio eletrônico; e *iv*) Sistemas de Informações Geográficas (SIG).

Atualmente, a utilização do termo ontologia é considerado comum na área de engenharia de software. De uma forma geral, na engenharia de software, uma ontologia pode ser vista como um modelo que representa conceitos e relações, bem como suas restrições, e pode ser definida como um artefato de engenharia. Alguns exemplos do uso da ontologia na engenharia de software são: processos de desenvolvimento software, qualidade do software, documentação, riscos e requisitos.

Ontologias são apontadas, hoje, como sendo cruciais na GC em melhoria dos processos. Muitos dos trabalhos que exploram o uso da GC e ontologia se baseiam no conceito de experiências passadas. Em um ambiente de desenvolvimento de software, por exemplo, o reuso das experiências e conhecimento coletados podem ajudar a evitar que falhas se repitam e auxilia na solução de problemas recorrentes do processo aplicado, promovendo sua melhoria.

2.3 Influência da GC na melhoria de processos de desenvolvimento de software

Pesquisas voltadas para a melhoria do processo de desenvolvimento software podem ser consideradas hoje uma das maiores prioridades para as organizações que trabalham com software. Diversos trabalhos surgiram ao longo dos últimos anos utilizando a GC com o propósito de entender e avaliar um processo de desenvolvimento de software, e somente então, tomar ações que apoiem a melhoria do processo. Alguns destes trabalhos foram selecionados para ilustrar as diferentes maneiras de lidar com tais temas e estão apresentados na Tabela 2.3.

Tabela 2.3 - Trabalhos relacionados: gestão de conhecimento, ontologia e melhoria de processos

ANO	DESCRIÇÃO
1998	No trabalho de Falbo (1998), é defendido o uso de Servidores de Conhecimento para promover a integração de conhecimento em Ambientes de Desenvolvimento de Software (ADSs). É apresentado um Servidor de Conhecimento que torna disponíveis componentes de conhecimento para serem reutilizados e compartilhados entre ferramentas. Sua arquitetura é projetada com base em ontologias e modelos de tarefa. O servidor foi definido no contexto do Projeto TABA desenvolvido no Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ.
2002	O Centro de Desenvolvimento de Sistemas de Vitória (CDSV), tem investido fortemente na qualidade dos processos de desenvolvimento de software. Em Borges e Falbo (2002), é apresentada uma ferramenta desenvolvida para apoiar a manutenção e melhoria contínua da qualidade do processo, apoiando a adaptação do processo padrão do CDSV para os projetos e permitindo o compartilhamento de conhecimentos adquiridos nesses projetos com a utilização da GC. A ferramenta, chamada de *ProKnowHow*, possui um repositório (repositório de experiências) contendo os conhecimentos formais e informais obtidos nos processos, organizados de forma a poderem ser reutilizados. Para construção do repositório foi utilizado uma ontologia de processo de software desenvolvida por Falbo (1998).

continua na próxima página

Tabela 2.3 - Trabalhos sobre gestão de conhecimento, ontologia e desenvolvimento de software

ANO	DESCRIÇÃO
2003	Falbo et al. (2003) propôs um ambiente denominado *Ontology-based software Development Environment* (ODE), a partir da necessidade de integrar ferramentas que pudessem oferecer suporte a todo o processo de desenvolvimento de software. Por ser um ambiente baseado em ontologias, o ODE possui um repositório de conhecimento, que proporciona ao ambiente uma uniformidade de conceitos, primordial na integração de ferramentas. O ambiente ODE vem sendo desenvolvido no Laboratório de Engenharia de Software - LabES - do Departamento de Informática da Universidade Federal do Espírito Santo (UFES). Foi implementado na linguagem Java, possuindo, assim, a característica de ser multiplataforma e a persistência é feita em um banco de dados relacional.
2006	Com o objetivo de evoluir a estrutura de processos de software do ODE, Bertollo (2006), estudou o estado da arte e o estado da prática de processos de software e propôs uma revisão na ontologia de forma a acompanhar esse novo contexto. A ferramenta de definição de processos de software de ODE também foi revisada, de forma a atender à nova conceituação.
2009	O trabalho de (MORO; FALBO, 2009) também está inserido no contexto do Projeto ODE, e tem por objetivo a definição de uma base conceitual para apoiar a automatização do processo de Avaliação e Melhoria de Processos de Software (AMP), baseado em uma ontologia para esse domínio. Outro objetivo específico do trabalho foi a definição de um processo de AMP, a partir do qual possam ser levantados requisitos funcionais para se prover apoio automatizado ao processo de AMP. Também foi desenvolvido ferramentas de apoio à AMP, tomando por base o processo definido e a ontologia de qualidade de software proposta. Duas ferramentas com funcionalidades básicas foram desenvolvidas: AvaliaODE, para apoiar a avaliação de processos e produtos de software em ODE, e MelhoriaODE, visando prover algum apoio a iniciativas de melhoria de processos em ODE.

continua na próxima página

Tabela 2.3 - Trabalhos sobre gestão de conhecimento, ontologia e desenvolvimento de software

ANO	DESCRIÇÃO
2010	Foi desenvolvida o *Software Measurement Ontology* (SMO), fundamentado na Ontologia Fundamental Unificada. Neste trabalho, (BARCELLOS et al., 2010), apresentou um fragmento do SMO com foco em software de análise de comportamento do processo. A ontologia foi criada com o objetivo de lidar com o problema da diversidade de vocabulário nos padrões e modelos de desenvolvimento utilizados no projeto de desenvolvimento de software.

Neste capítulo, foi apresentada brevemente conceitos sobre GC e ontologia, e uma síntese das principais pesquisas que estão sendo conduzidas referente à melhoria de processo de desenvolvimento de software. Por meio da revisão realizada até o presente momento, podem-se identificar diversas linhas de pesquisa ainda em aberto e que merecem investigação.

Diversos trabalhos apresentam como a GC tem influenciado na melhoria de processos de software, mas vale destacar que a maior parte das pesquisas estão voltadas para melhoria do processo de desenvolvimento de software. Pouca atenção é dada para os trabalhos que envolvem tais tecnologias no ambiente de teste de software com o propósito de promover a melhoria dos processos de teste.

Dada a carência de trabalhos na área, ao mesmo tempo, a relevância do tema, identificou-se a importância de se definir estratégias e mecanismos de apoio à melhoria dos processos de teste de software a partir dos princípios da GC. Mas para discutir sobre melhoria de processos de teste é necessário conhecer alguns conceitos básicos sobre o tema. No próximo capítulo são apresentados brevemente os principais conceitos referentes à atividade de teste, o que tem sido realizado na literatura como melhoria desses processos e também como a GC pode ser aplicada no ambiente de teste de software.

3 TESTE DE SOFTWARE

3.1 Terminologia e conceitos

Para o melhor entendimento deste trabalho, é necessário abordar brevemente alguns conceitos importantes relacionados à atividade de teste de software. Testes de software são atividades realizadas em um projeto com o objetivo principal de contribuir com a qualidade dos produtos gerados. Muitas definições de qualidade de software são propostas na literatura. De acordo com a Norma Técnica de Sistemas de Gestão da Qualidade (ABNT NBR ISO 9000)(ISO9000, 2006), "a qualidade é o grau no qual um conjunto de características inerentes satisfaz aos requisitos". Ou seja, pode-se afirmar que se algum produto ou serviço atende aos requisitos especificados, este mesmo produto ou serviço possui, em princípio, a qualidade desejada.

Verificação e Validação são dois importantes conceitos relacionados à qualidade do software. A Norma *Institute of Electrical and Electronics Engineers* (IEEE) 610.12-1990 (IEEE, 1990), faz as seguintes definições:

a) Verificação: é o processo de avaliação de um sistema ou componente para determinar se os produtos de uma determinada fase de desenvolvimento satisfazem as condições impostas no início dessa fase; e

b) Validação: é o processo de avaliação de um sistema ou componente durante ou no final do processo de desenvolvimento para determinar se satisfaz os requisitos especificados.

Outros três importantes conceitos relacionados à área de teste de software são: Defeito, Erro e Falha. Na área de teste de software, existem divergências quanto à terminologia adotada. Neste trabalho, é utilizada a definição estabelecida pelo IEEE 610.12-1990 (IEEE, 1990), que se esforça para padronizar a terminologia utilizada em vários campos do conhecimento, dentre eles o da engenharia de software. A norma IEEE distingue os termos da seguinte forma:

a) Defeito *(fault)*: é um passo errado, processo ou definição de dados em um programa de computador;

b) Erro *(error)*: é a manifestação concreta de um defeito num artefato do software, ou seja, é a diferença entre valor obtido e o valor esperado; e

c) Falha *(failure)*: é o comportamento operacional do software diferente do esperado pelo usuário.

Existem atualmente várias definições para o conceito de teste de software. De uma forma simples, testar um software significa avaliar, através de uma execução controlada, se o seu comportamento ocorre de acordo com o que foi especificado. O principal objetivo do teste é revelar a presença de erros no produto (MYERS, 2004). Nesse sentido, uma atividade de teste pode ser considerada como bem sucedida quando o programa em teste falha (PRESSMAN, 2006).

Até a década de 90, os testes eram realizados pelos próprios desenvolvedores do software e foi considerado uma metodologia inapropriada. As informações extraídas desses testes não permitiam que todos os defeitos fossem detectados. A situação piorou quando começaram a surgir aplicações muito mais complexas, devido ao surgimento de novas tecnologias, levando as organizações a procurarem novas soluções para melhorar a qualidade do software desenvolvido. Foi a partir desta década que o teste de software começou a ser tratado não mais como uma atividade do processo de desenvolvimento, e sim como um processo independente (BASTOS et al., 2007).

O objetivo principal do processo de teste, com metodologia própria, é minimizar os riscos causados por defeitos provenientes do processo de desenvolvimento e agregar qualidade ao produto gerado. Segundo Delamaro et al. (2007), a Validação, Verificação e o Teste, ou "VV&T", tem como finalidade garantir que tanto o modelo pelo qual o software é construído quanto o produto em si estejam em conformidade com o especificado. As atividades de "VV&T" estão distribuídas nas diversas etapas do processo de teste (BASTOS et al., 2007).

3.2 Processo de Teste de Software

Um processo de desenvolvimento de software é um conjunto de ferramentas, métodos e práticas usado para produzir software. Tal processo é representado por atividades sequenciais que integram estratégias para cumprimento da evolução do software (PRESSMAN, 2006). No processo de desenvolvimento, as etapas básicas que o compõe são: análise, projeto, implementação, teste, implantação e manutenção. A atividade de teste, pela sua importância na qualificação de um software, também é tratada como um processo e possui etapas similares ao processo de desenvolvimento de software.

Segundo Bastos et al. (2007), o ciclo de vida de testes pressupõe que sejam realizados testes ao longo de todo o processo de desenvolvimento de software. Os processos de desenvolvimento e teste têm início simultâneo, ou seja, a equipe que desenvolve o software inicia o processo de desenvolvimento, e a equipe que está conduzindo os testes começa o planejamento do processo de teste. Ambas as equipes começam no mesmo ponto, usando as mesmas informações. Esse comportamento é chamado de conceito "V" e é ilustrado na Figura 3.1. No conceito "V" de teste, os procedimento de "VV&T" convergem do início até o fim do projeto.

Figura 3.1 - Conceito "V" de teste de software.
Fonte: Bastos et al. (2007)

O processo de teste é composto por diversas etapas ou fases. De acordo com Rios e Moreira (2006), o processo de teste é composto por quatro fases sequenciais (Procedimentos Iniciais, Especificação, Execução e Entrega), e duas paralelas (Planejamento e Preparação), mostrado na Figura 3.2. O modelo foi chamando por Rios e Moreira (2006) de Modelo 3Px3E. Cada uma das etapas é composta por atividades, produtos e documentos.

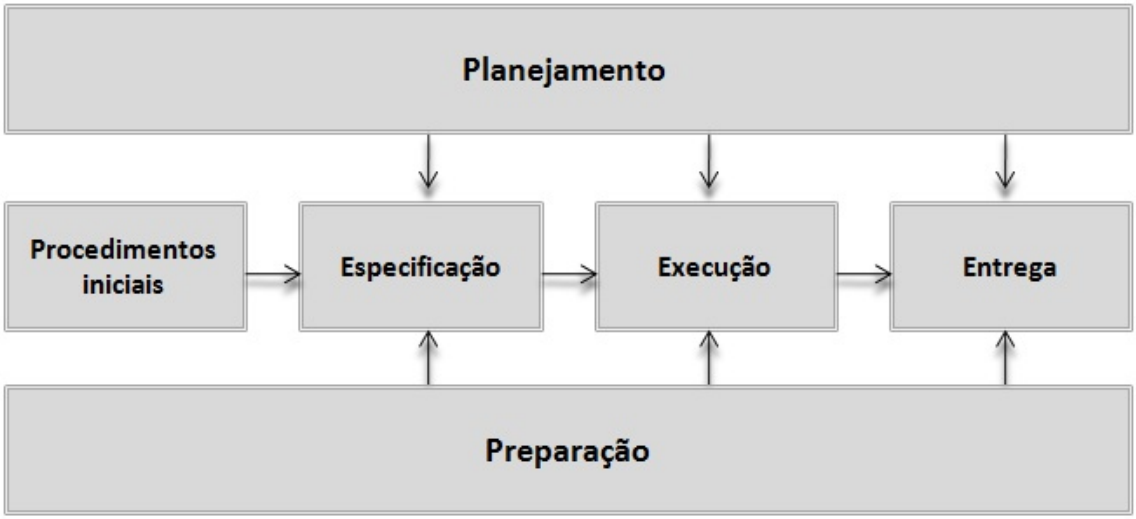

Figura 3.2 - Modelo 3Px3E do ciclo de vida do processo de teste.
Fonte: Rios e Moreira (2006)

- **Procedimentos Iniciais**: são condutas estabelecidas e ordenadas para a realização de atividades. É a etapa na qual é realizado um estudo dos requisitos do negócio e elaborado um pequeno esboço do processo de teste.
- **Planejamento**: caracteriza-se pela definição de uma proposta de teste (Plano de Teste), apresentando prazos e esforços de teste a serem realizados.
- **Preparar**: finalizado o planejamento, deve-se preparar o ambiente de teste (atividades, equipamentos, pessoas, ferramentas de automação, hardware e software).
- **Especificação**: é uma etapa que é caracterizada pela identificação dos casos de teste[1] que deverão ser construídos.
- **Execução**: os casos de teste gerados na especificação são exercitados.
- **Entrega**: entrega de artefatos com a avaliação de todo o processo de teste do software, comparando os resultados alcançados em relação ao que foi inicialmente planejado.

[1]Conjunto de entradas de teste, condições de execução e os resultados esperados desenvolvidos para um objetivo específico, como de exercer um caminho determinado programa ou verificar a conformidade com um determinado requisito (IEEE, 1990).

A única maneira de se garantir a correção de um software seria através de um teste exaustivo, executando-se o software com todas as combinações de valores de entrada. Esta prática é inviável, pois o domínio de entrada pode ser infinito ou, no mínimo, muito grande (MYERS, 2004). Portanto, devem ser utilizadas técnicas para selecionar um subconjunto de dados de teste. Cada etapa de teste é realizada mediante uma série de técnicas de teste e diversos critérios pertencentes a cada uma delas. As técnicas de teste são classificadas de acordo com a origem das informações utilizadas para estabelecer os requisitos de teste (DELAMARO et al., 2007). As três principais técnicas de teste utilizadas são: Teste Estrutural (ou caixa-branca), Teste Funcional (ou caixa-preta) e Teste Baseado em Defeitos (DELAMARO et al., 2007).

A Técnica Estrutural tem como objetivo testar a estrutura de controle do programa. Os caminhos lógicos do programa são testados, fornecendo casos de teste que põem à prova tanto conjuntos específicos de condições e laços, bem como definições e usos de variáveis. No caso da Técnica Funcional, são fornecidas entradas e avaliadas as saídas geradas para verificar se estão em conformidade com o esperado. Nesta técnica os detalhes de implementação não são considerados. Já a Técnica Baseado em Defeitos utiliza-se de informações sobre os defeitos mais frequentes que podem estar presentes em um produto para derivar casos de teste. A Figura 3.3 apresenta a relação que existe entre as principais técnicas, critérios e níveis de teste.

Considerar o processo de teste uma atividade que deve ser controlada, medida e melhorada é um passo importante para torná-lo realmente efetivo. Para tanto, é indispensável que se tenha o controle do relacionamento entre as atividades, as ferramentas, o pessoal, os métodos e principalmente as características do produto de software que é o objetivo final. De acordo com Bastos et al. (2007), a qualidade do produto está relacionada com a qualidade do processo de teste, pois o processo define como os testes serão planejados e exercitados, então o processo deve ser claramente definido.

3.3 Qualidade dos Processos de Teste

Conforme apresentado na seção 3.2, a atividade de teste já é tratada como um processo independente que apresenta índices de qualidade melhores para os produtos. No entanto, essa mudança organizacional trouxe novos problemas a serem resolvidos. Não adianta apenas testar, mas sim testar bem (RIOS, 2009). Se a área de teste não estiver bem organizada os defeitos vão continuar a ocorrer num estágio onde os custos

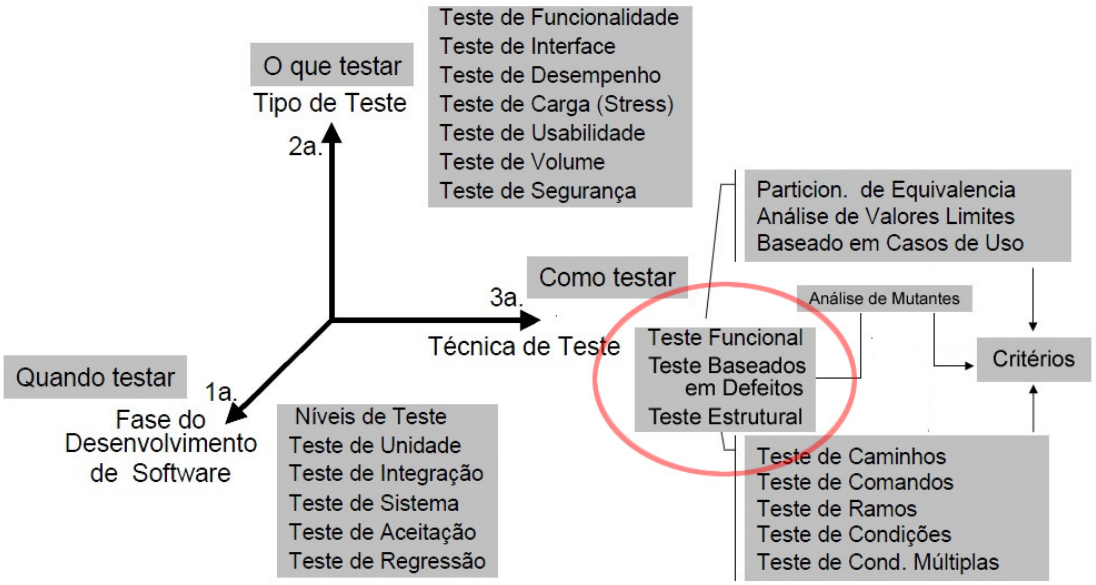

Figura 3.3 - Relação entre níveis, técnicas e critérios de teste.
Fonte: Adaptado de Crespo et al. (2004)

acabam se tornando maiores prejudicando o orçamento de todo o desenvolvimento.

A área de melhoria dos processos é investigada sistematicamente, dadas as evidências de que a qualidade do processo pode influenciar significativamente na qualidade do produto final. Nesse sentido, a melhoria dos processos é um desafio importante para as organizações. Segundo Koomen e Pol (1999), o conceito de melhoria do processo pode ser definido como sendo a otimização dos custos da qualidade e prazo de execução dos processos, em relação ao total serviços de informação.

Diversas abordagens são desenvolvidas ao longo dos anos para a implantação de métodos e modelos para melhoraria de processos. A maioria destas abordagens surgiram como melhoria do processo de desenvolvimento do software, e atualmente estão sendo empregadas também na melhoria dos processos de teste. Tais abordagens, apresentadas brevemente a seguir, são utilizadas como guias para implantar melhoria contínua aos processos, pois a partir de repetições de um conjunto de atividades (ciclos), tem como objetivo aperfeiçoar o processo a cada novo ciclo.

- **Ciclo *Plan/Do/Check/Action* (PDCA)**: O ciclo PDCA, proposto por Walter Shewart e divulgado por Deming (1986), é direcionado a aplicar um

ciclo de melhoria no processo. O ciclo tem por princípio tornar mais claros e ágeis os processos envolvidos na execução da gestão, como a gestão da qualidade, dividindo-a em quatro principais fases: Planejamento, Execução, Verificação e Ação.

A fase de **Planejamento** (*Plan*) estabelece um plano de melhoria dos processos existentes, definindo metas e métodos para atingi-la. A fase de **Execução** (*do*) executa o trabalho planejado e coleta os dados de sua execução. Na fase de **Verificação** (*Check*) é comparado o resultado do trabalho em cada estágio de execução com os critérios e metas estabelecidas na fase de Planejamento. Por fim, é na fase de **Ação** (*Action*) que os resultados obtidos são estudados, identificando o que foi aprendido e quais as melhorias que podem ser aplicadas no próximo ciclo.

- **Método *Goals Questions (Indicators) Measures* (GQ(I)M)**: O método GQ(I)M (PARK et al., 1996) é uma evolução do método inicial *Goal Questions Measures* (GQM), proposto por (BASILI et al., 1994). O método GQM descreve o planejamento e execução de um programa de mensuração de produtos e processos de software. O que distingue GQ(I)M da metodologia GQM são os indicadores (I). Os indicadores, provenientes de métricas, fornecem uma compreensão subjetiva do *status* de uma área designada do sistema que serão utilizados como calibradores em todo o processo.

- **Modelo IDEAL**: O IDEAL (GREMBA; MYERS, 1997) é um modelo de ciclo de vida para a melhoria de processo, definido pela *Software Engineering Institute* (SEI) e baseado no *Capability Maturity Model* (CMM). O modelo descreve cinco fases: *i*) Início (*Initiating*): definição do objetivo e infraestrutura para alcançá-lo; *ii*) Diagnóstico (*Diagnosing*): determinar a situação atual em relação à situação desejada, com base no objetivo almejado; *iii*) Planejamento (*Establishing*): planejar os passos necessários para alcançar o objetivo; *iv*) Ação (*Acting*): realizar os passos especificados no planejamento, pesquisando soluções para problemas e implementando melhorias em toda organização; e *v*) Aprendizado (*Learning*): aprender com a experiência e melhorar mudanças futuras.

Atualmente, a grande maioria das abordagens e experiências encontradas na literatura, para a melhoria de processo de software são baseadas em modelos de maturi-

dade (MPS.BR, 2005) e são utilizados com sucesso nas instituições acadêmicas e na própria indústria. Os modelos de maturidade fornecem aos gestores um poderoso instrumento para avaliarem os processos utilizados na organização. Por esse motivo, é apresentada na próxima seção uma descrição mais detalhada sobre os modelos de maturidade e alguns modelos voltados especificamente para os processos de teste.

3.4 Modelos de Maturidade

Diversos modelos de maturidade foram propostos ao longo dos anos, e são usados como base para avaliar diferentes organizações e estabelecer comparações. Estes modelos diferem sobretudo no número de etapas, variáveis de evolução e áreas de foco. Cada modelo identifica certas características que tipificam o alvo em diferentes níveis de maturidade.

Os modelos descrevem a maturidade da empresa baseado nos projetos que ela está desenvolvendo. A maturidade implica num potencial de crescimento da capacidade e indica tanto a riqueza do processo na organização, quanto a consistência na qual o processo é aplicado nos projetos (PAULK, 1995; PAULK, 1998).

As décadas de 80 e 90 marcaram o surgimento dos grupos de garantia de qualidade na área de Tecnologia de Informação (TI). Para melhorar o processo de desenvolvimento de software, vários modelos de maturidade foram criados e hoje servem como diretrizes para muitas organizações. Entre os modelos que mais se destacam têm-se o CMM (PAULK, 1995) que é um modelo largamente usado como padrão para melhoria de processos de software em indústrias. O CMM foi projetado para auxiliar na escolha das políticas (o que fazer) e nas estratégias (como fazer). Outro modelo que se destaca é o CMMI (SEI, 2006), criado pela SEI, com o objetivo de integrar os diversos modelos do CMM existentes.

No Brasil foi criado o modelo MPS.BR (MPS.BR, 2005). O MPS.BR é um modelo de qualidade de processo voltada para a realidade do mercado de pequenas e médias empresas de desenvolvimento de software no Brasil. Ele é baseado no CMMI e na realidade do mercado brasileiro.

Embora os modelos de maturidade de software existentes tenham grande aceitação pelas organizações de desenvolvimento de software, eles não abordam adequadamente as questões relacionadas às atividades de teste e nem possuem um processo de teste bem definido (RIOS; CRISTALLI, 2007). Com isso, surgiram pesquisas volta-

das para criação de modelos de maturidade de teste que completassem as deficiências na área de teste.

3.4.1 Modelos de Maturidade para Processo de Teste

Diversos trabalhos surgiram com o intuito de caracterizar as necessidades de um processo de teste e avaliá-los com o objetivo de prover melhoria. Atualmente existem diversos modelos de avaliação da maturidade dos processos de teste. Os mais conhecidos são:

- ***Test Improvement Model*** (TIM)

  O TIM foi baseado no CMM e possui a intenção de guiar funções de teste de software no seu processo de melhoria. Segundo Ericson et al. (1997), existem dois propósitos para seu uso: i) a identificação do estado atual das práticas das áreas chaves de processo (*Key Process Areas*), ou seja, identificação dos pontos fortes e fracos; e ii) a sugestão de boas práticas para a implementação dos pontos fortes e remoção dos pontos fracos.

- ***Test Maturity Model*** (TMM)

  Criado pelo *Illinois Institute of Technology* (BURNSTEIN et al., 1996), o TMM foi baseado no CMM com o qual mantém compatibilidade. O modelo é baseado na avaliação da situação atual do processo de teste através de regras claras e objetivas. O TMM almeja a avaliação de um processo de teste já existente. O propósito do TMM está relacionado nas seguintes afirmativas: deve ser uma forma de avaliação interna da equipe para identificar o seu correto estado de capacitação; dever ser claro para que as gerências superiores possam aceitá-lo como um programa de melhoria de teste; deve permitir que as equipes de desenvolvimento possam também melhorar os seus testes; e os usuários e clientes devem também poder ocupar o seu papel no processo de melhoria.

- ***Test Process Improvement*** (TPI)

  O TPI foi desenvolvido com base na prática, conhecimentos e experiências adquiridos a partir do desenvolvimento do processo de teste (ANDERSIN, 2004). É considerado um método para caracterização de um processo de teste e identificação de oportunidade de melhorias, usando um modelo estático (CRAIG; JASKIEL, 2002). Assim como o TMM, o modelo TPI somente

almeja a avaliação de um processo de teste já existente. Este modelo permite aos seus usuários determinar o estado atual do seu processo de teste, através de diversas áreas chaves do processo, semelhante ao que é realizado no CMM.

- **Modelo Brasileiro Melhoria de Processo de Teste (MPT.BR)**

  O modelo brasileiro de Melhoria de Processo de Teste (MPT.BR) encontra-se ainda em desenvolvimento. O MPT.BR é baseado nas melhores práticas do teste de software e promove a integração das atividades de engenharia de software. O MPT.BR também baseia-se no modelo MPS.BR e em alguns critérios usados pelo CMMI (MPT.BR, 2009). O principal objetivo do MPT.BR é garantir que áreas de teste de software de tamanho reduzido possam ser avaliados e estimulados a alcançar níveis maiores de maturidade, sem que para isso tenham que incorrer em altos custos operacionais.

A comparação entre os modelos internacionais TMM, TIM e TPI, e o modelo brasileiro MPT.BR, pode ser visualizada na Tabela 3.1.

Tabela 3.1 - Comparação entre os modelos TMM, TIM, TPI e MPT.BR. Adaptado de (SWINKELS, 2000; MPT.BR, 2009) .

	TMM	**TIM**	**TPI**	**MPT.BR**
Tipo Modelo	Maturidade	Maturidade	Maturidade	Maturidade
Ano de desenv.	1996	1996	1997	2009
Abordagem	Conceitual	Prática	Prática	Prática
Níveis	5	5	14	7
Key Process Area (KPAs)	13	5	20	16
Tipo de avaliação	Questionário	Questionário	*Checklist*	Questionário e *Checklist*
Elementos de avaliação	Procedimento de avaliação, Questionário	Orientação de avaliação, Questionário	Orientação de avaliação, *Checklist*	Comprovação objetiva e Exame das evidências

continua na próxima página

Tabela 3.1 - Comparação entre os modelos TMM, TIM, TPI e MPT.BR

	TMM	**TIM**	**TPI**	**MPT.BR**
Informações sobre o modelo	Artigos e Dissertações	Artigos	Artigos e livros	Artigos e Dissertações

3.5 GC na Melhoria de Processos de Teste

A comunidade científica desenvolve pesquisas buscando garantir a qualidade do produto de software. Tais pesquisas indicam que realmente a qualidade do produto de software é fortemente dependente da qualidade dos processos que fazem parte do projeto. Diante desse contexto, as organizações podem promover a melhoria de um processo de desenvolvimento de software, por meio do aprimoramento da maturidade da organização, conforme apresentado na seção anterior (seção 3.3).

Na área de teste de software, por ser um fator crítico na qualidade do produto, observa-se uma preocupação maior com pesquisas relacionadas ao tema. Lamas (2010), propôs o detalhamento de uma estrutura de maturidade operacional para gestão de testes de software, denominado *Organizational Testing Management Maturity Model* (OTM3) (LAMAS et al., 2010). A estrutura visa a aferição de resultados de teste de software, através de um método que utiliza medidas de eficiência e eficácia, estruturadas no método GQ(I)M. Isso permite identificar, estabelecer e manter as capacidades exigidas por Modelos de Maturidade de Teste de Software. A atualização do OTM3 é apoiada com uma avaliação contínua pelo ciclo de PDCA. Foi utilizado como estudo de caso o projeto de Integração e Cooperação Amazônica para Modernização do Monitoramento Hidrológico (ICA-MMH) apoiado pela Financiadora de Estudos e Projetos (FINEP), Projeto 5206/06 (CUNHA, 2010).

No projeto ICA-MMH, foi gerado um conjunto significativo de dados de teste a partir do processo de teste que segue a estrutura do OTM3. Para análise desses dados, no trabalho de Ferreira (2010), foram utilizadas técnicas de Mineração de Dados (FAYYAD et al., 1996), afim de levantar informações relevantes sobre do processo de teste empregando o algoritmo de associação Apriori (AGRAWAL et al., 1993). O objetivo era a identificação de regras ou correlação entre os dados de teste, tais como, resultados dos casos de teste exercitados, gravidade do defeito, prioridade

de correção, dentre outros. O trabalho mostrou como resultados preliminares, a coerência existente entre os padrões encontrados e o processo de teste, e também mostrou ser um bom ponto de partida para realizar estudos semelhantes.

Apesar de não ter sido o escopo inicial, o trabalho de Ferreira (2010) mostrou a possibilidade de trabalhar os dados gerados em um ambiente de teste de software com o objetivo de encontrar informações importantes para melhoria do processo de teste. Porém, encontrar informações relevantes (conhecimento) pode ser uma tarefa árdua e complexa, e está relacionado, sobretudo, à falta de semântica associada ao grande volume de informações. A partir das pesquisas realizadas, percebeu-se que o emprego da GC pode ser uma boa estratégia para lidar com a situação apresentada anteriormente. O contexto apresentado também serviu como motivação para a pesquisa dos temas abordados e elaboração desta proposta de trabalho de pesquisa.

Conforme apresentado (seção 3.3), os modelos de maturidade avaliam a estrutura e controle organizacional dos projetos. Tais modelos funcionam como um roteiro para a melhoria dos processos de forma organizacional. A comunidade acadêmica também tem trabalhado em dados coletados ao longo dos projetos, que podem fornecer informações importantes para promover ações corretivas e boas práticas em projetos futuros. No entanto, mesmo sendo um fator crítico de qualidade do produto, o processo de teste compõe-se de poucas pesquisas neste sentido. A literatura ainda carece de trabalhos que estabeleçam estratégias de melhorias no processo de teste para serem usadas na tomada de decisões a partir de princípios da GC.

De acordo com Wu e Xuemei (2009), a aplicação da GC de forma efetiva no processo de teste de software, é a chave para melhorar a qualidade dos testes realizados. Wu e Xuemei (2009) propôs um modelo de GC para processo de teste de software. Para propor esse modelo, o autor definiu cinco problemas existentes em um processo de teste: *i*) pouca reutilização do conhecimento; *ii*) barreiras em relação ao compartilhamento de informação; *iii*) estrutura do ambiente; iv) perda de conhecimento (experiências e habilidades) de integrantes da equipe; e *v*) distribuição dos recursos humanos. A ideia do modelo é construir uma base de experiências relatadas pelos membros da equipe de teste. É utilizado como estudo de caso um subsistema do sistema QESuite2.0 desenvolvido por *Beijing University of Aeronautics and Astronautics* (BUAA) (GU et al., 2008).

O trabalho de Desai (2011) descreve a reutilização do conhecimento na área de teste

de software. Módulos, casos de teste ou componentes reutilizáveis são armazenados em uma biblioteca de componentes para ser referenciados usando mecanismos de busca. A GC ajuda a construir, armazenar e manter esta biblioteca de componentes de software. Estes dados são armazenados na biblioteca, de tal forma que podem ser referenciados a partir de qualquer lugar ao redor do mundo e podem ser atualizados conforme a utilização pela *Internet*. Desai (2011) também destaca vários problemas enfrentados pelas organizações em relação à atividade de teste de software, e a falta de soluções para estes problemas, um deles é a perda de conhecimento que ocorre quando um indivíduo experiente deixa a organização e cita a GC como uma possível solução.

Segundo Wu e Xuemei (2009), é difícil encontrar trabalhos que abordam a GC aplicada no campo de teste de software. Até o momento trabalhos com soluções eficientes com GC e teste de software, em especial processos de teste, não foram encontrados, mas existem trabalhos relacionados na GC em outros campos, como os citados anteriormente. Wu e Xuemei (2009) também descrevem em seus trabalhos, que anualmente acontece o *Knowledge-Based Software Engineering Conference* (KBSE), em qual o mais recente avanço da GC em teste de software é discutido.

No Brasil, foram encontrados alguns trabalhos com finalidades distintas, mas que estão relacionados à construção de ontologias em ambientes de teste de software. No trabalho de Barbosa (2004), são discutidos e estabelecidos mecanismos de apoio à atividade de modelagem de conteúdo e ao processo de desenvolvimento de módulos educacionais na Engenharia de Software. A construção desses módulos educacionais pode ser investigada sob quatro perspectivas correlatas: estruturação, transferência, evolução e **reuso do conhecimento**. Dessa forma, é feito um discurso da criação de ontologia na área de engenharia de software, em particular com atividades de teste de software. Outros trabalhos surgiram nessa mesma linha, tais como Barbosa et al. (2006), Barbosa et al. (2008) e Nakagawa et al. (2009).

Apesar da literatura conter trabalhos que abordam os temas que serão trabalhos nesta proposta, algumas questões continuam em aberto, principalmente questões relacionadas à melhoria dos processos de teste de software com base nos princípios da GC. Diante do contexto exposto, com o interesse de envolver pesquisas voltadas para criação de estratégias de melhoria de processos de teste de software e com a intenção de suprir as lacunas apontadas, é apresentado no próximo capítulo a proposta de trabalho de pesquisa.

4 ESTRATÉGIA PARA POSSIBILITAR A MELHORIA DE PROCESSO DE TESTE

Foi estudada, nas seções anteriores, a influência da GC e da ontologia na qualidade do software, principalmente como estas tecnologias são aplicadas na melhoria de processos. A grande maioria dos trabalhos encontrados na literatura que utiliza GC como ferramenta para melhoria dos processos, são direcionados para os processos de desenvolvimento de software. O processo de teste compõe-se de poucas pesquisas para sua melhoria. Isto é, há necessidade de envolver pesquisas voltadas para criação de mecanismos que possam promover uma melhoria de processos de teste de software.

Conforme mencionado antes, o processo de teste gera um grande volume de informações. Por esse motivo, nem sempre é possível encontrar facilmente correlação entre as informações, e gerar conhecimento adequado para determinar ações de melhoria e reutilização deste conhecimento. O atual desafio é conceber soluções que ofereçam suporte na melhoria de processos de teste de software, permitindo o armazenamento, o compartilhamento e o reuso do conhecimento nestes ambientes.

4.1 Concepção e Desenvolvimento

Uma das técnicas mais utilizadas para proporcionar esse entendimento sobre o conhecimento é o uso dos princípios da GC, principalmente se conciliada com a estrutura de um modelo que propicie o armazenamento, o compartilhamento e o reuso do conhecimento, conforme apresentado na Figura 4.1.

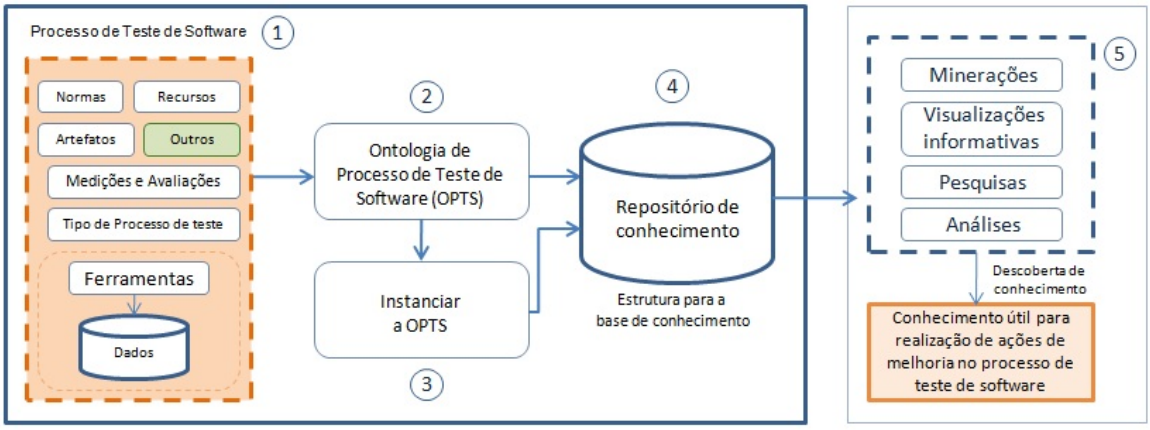

Figura 4.1 - Estrutura de um modelo de descoberta de conhecimento.

Considerando a estrutura proposta apresentada acima, será apresentado, em linhas gerais, cada objetivo específico da proposta de trabalho de pesquisa, de acordo com a enumeração ilustrada na própria figura.

1 - Caracterizar o processo de teste de software

O primeiro objetivo é caracterizar o processo de teste de software. É importante enfatizar que organizações que desenvolvem software lidam com diversos tipos de projetos e não existe um processo que seja adequado às particularidades de todo e qualquer projeto. Neste sentido, o primeiro aspecto é identificar características comuns a estes projetos. Em função das características específicas de um determinado domínio, um processo padrão pode ser definido. Um processo padrão é o conjunto de elementos fundamentais que guia o estabelecimento de um processo comum na organização (BORGES; FALBO, 2002). A definição de um processo padrão ajudará nas próximas etapas do trabalho, por exemplo, na criação da ontologia.

Outra característica importante que deve ser trabalhada, é a utilização, nos projetos, de diversas normas e modelos de qualidade de processo distintos (seção 3.3). Cada norma ou modelo, define os seus próprios conceitos e trabalha de forma independente. Não há um vocabulário comum compartilhado. Neste contexto, ontologias se apresentam como candidatos potenciais e úteis, pois podem ser usadas como uma estrutura unificadora para dar semântica e uma representação comum à informação (FALBO et al., 2004b).

Um processo de teste não se caracteriza apenas pela sua especificidade para cada projeto de teste, normas e padrões. É importante considerar ferramentas de gestão de testes e os dados gerados por estas ferramentas. Normalmente os dados de relatos de teste encontram-se armazenados em bases de dados gerados por ferramentas de gestão muito utilizadas em ambientes de teste. Exemplos de ferramentas livres usadas para o gerenciamento da atividade de teste são: *Mantis* (MANTIS, 2011) e *TestLink* (TESTLINK, 2011). Mesmo empregando uso de ferramentas, conforme a evolução da proposta deste trabalho, pode surgir a necessidade de obter algum tipo de informação que não é possível encontrar em tais ferramentas. Dessa forma, uma possibilidade é o uso de questionários, os quais podem conter, por exemplo, uma avaliação qualitativa para composição de métricas de testabilidade.

2 - Criar uma ontologia de processo de teste de software

O segundo objetivo, que certamente demandará um maior esforço para desenvolver esta proposta, é criar uma Ontologia de Processo de Teste de Software (OPTS) que organize e estruture o conhecimento gerado a partir da caracterização do domínio. A definição de um vocabulário comum ao domínio de processos de teste de software que represente o conhecimento relevante a esse domínio contribui para o compartilhamento e a reutilização desse conhecimento, bem como para a interoperabilidade semântica.

Para construir uma ontologia é necessário utilizar uma metodologia que apoie na sistematização dessa tarefa. Esta metodologia deve também possibilitar a sua manipulação. Conforme apresentado na seção 2.2.1, existem diversos métodos para construção de uma ontologia. No entanto, Falbo (1998) reuniu em uma única proposta, as melhores características dos métodos existentes, definindo as seguintes etapas: *i*) Propósito e Especificação de Requisitos; *ii*) Captura da Ontologia; *iii*) Formalização da Ontologia; *iv*) Integração de Ontologias Existentes; *v*) Avaliação da Ontologia; e *vi*) Documentação da Ontologia. Pretende-se usar neste trabalho de pesquisa o método proposto por Falbo (1998), Falbo (2004a), chamado método SABiO.

Na linha do trabalho de Falbo (1998), há um trabalho que foca mais na etapa *iv* do método SABiO. Para construir a ontologia OPTS, alguns conceitos relacionados a outras ontologias já criadas podem ser requeridas. Buscando reutilizar conceituações previamente definidas, poderão ser utilizados conceitos originalmente estabelecidos por (BARBOSA, 2004; BARBOSA et al., 2006) (seção 3.5). Acredita-se que a proposta discutida aqui possa se beneficiar de alguns conceitos descritos por tais trabalhos, mesmo estes não sendo direcionados para melhoria de processos de teste de software.

Segundo Falbo (1998), a construção de uma ontologia deve ser vista como um processo iterativo, e não como passos sequenciais. Cada uma das etapas do método de construção pode ser retornada e revista. As etapas do processo de desenvolvimento de um ontologia e suas interdependências são ilustradas pela Figura 4.2.

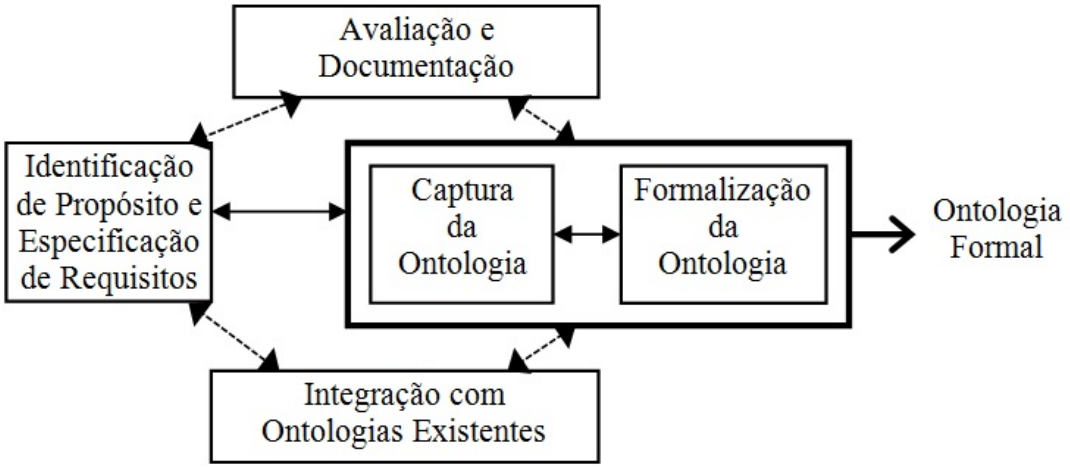

Figura 4.2 - Etapas do Desenvolvimento de uma Ontologia e suas Interdependências. Fonte: Falbo (1998)

A fase final no desenvolvimento de uma ontologia é realizada por meio de uma representação específica que permite o processamento e a abrangência do conhecimento. Isso é possível através de uma linguagem específica para a criação de ontologias e de uma ferramenta que permita sistematizar e integrar as especificações definidas à linguagem utilizada. A linguagem utilizada neste trabalho será a OWL, linguagem recomendada pela W3C e, para desenvolver a ontologia através da utilização desta linguagem, será utilizada a ferramenta Protégé. A ferramenta trabalha com várias linguagem inclusive a OWL.

A construção da ontologia OPTS também pode ir em direção ao desenvolvimento de um ambiente de ontologia para teste de software, ou contribuir com um ambiente já existente, como o ambiente ODE (FALBO et al., 2003). Algumas das ontologias que compõem a base ontológica do ODE são as seguintes: Ontologia de Processo de Software (FALBO; BERTOLLO, 2005); Ontologia de Qualidade de Software (DUARTE; FALBO, 2000); Ontologia de Artefatos de Software (NUNES, 2005); Ontologia de Gerência de Configuração de Software (GCS) (NUNES, 2005); Ontologia de Requisitos de Software (NARDI; FALBO, 2006); e Ontologia de Risco (FALBO et al., 2004c).

3 - Definir instâncias para a ontologia

Após criar a ontologia OPTS, definem-se as instâncias para a base da ontologia. As instâncias representam indivíduos específicos de um determinado domínio. Uma instanciação de uma ontologia consiste de declarações sobre objetos do domínio de discurso, usando os conceitos e as relações que foram definidas na ontologia (VALENTE, 1995). Estas declarações, juntamente com os axiomas da ontologia, formam a base de conhecimento reutilizável.

As instâncias criadas a partir da ontologia definida permitem fazer algum tipo de consulta já que há uma base delas. Estas consultas podem ser importantes para inferir algum tipo de informação. Atualmente, existem muitas opções de ambientes que fazem a interação do usuário com as informações da ontologia. A *Protocol and RDF Query Language* (SPARQL) (SPARQL, 2008), por exemplo, é uma linguagem recomendada pela W3C para consulta à RDF. É uma linguagem orientada a dados, que recupera informações contidas em arquivos RDF, possibilitando inclusive a opção de combinar dados de arquivos de diferentes fontes.

Existe também a linguagem *Semantic Query-enhanced Web Rule Language* (SQWRL) (MARTIN; AMAR, 2008), que é mais simples e expressiva, e é destinada para a realização de consultas em ontologias OWL. Ambas as linguagens estão incluídas na ferramenta Protégé. Nesta proposta de trabalho de pesquisa, pretende-se também investigar métodos de consultas, a fim de demonstrar possíveis consultas que podem ser realizadas na ontologia criada.

4 - Estruturar uma base de conhecimento

As ontologias são usadas para guiar a aquisição do conhecimento específico de uma organização. Também podem diminuir os custos da aquisição, permitir reutilização e compartilhamento de conhecimento (FALBO, 1998). Dessa forma, o quarto objetivo desta proposta de pesquisa de doutorado, é estruturar uma base de conhecimento para armazenar a ontologia criada e suas instanciações.

A ontologia criada neste trabalho ajudará a capturar o conhecimento sobre o domínio de processos de teste e tornar este conhecimento disponível em uma estrutura viável. A intenção da base de conhecimento é apoiar o armazenamento adequado dos dados em uma estrutura que possibilite a manipulação dos mesmos, como pesquisas,

minerações e análises.

Uma vez criada a base de conhecimento, podem ser realizados diversos experimentos promovendo um entendimento fácil aos especialistas do domínio sobre o processo, podendo tirar melhor proveito das interpretações e identificação de oportunidades de melhorias do processo de teste.

É necessário salientar que, ainda não foi possível definir qual melhor estrutura a ser seguida para criar a base de conhecimento. Dependendo do volume de informações, teria que ser avaliado se esta base de conhecimento será uma tabela, um banco de dados ou estruturas mais complexas como *Data Marts* ou *Data Warehouse*.

5 - Descobrir conhecimento novo e analisar os benefícios e as oportunidades de melhoria para a estratégia proposta

Realizar testes de software não consiste simplesmente na geração e execução de casos de teste, mas envolve também questões de planejamento, gerenciamento e principalmente a análise dos resultados. A fim de conhecer e identificar mecanismos eficientes para dar suporte a tomada de decisão que trabalhem com dados de processo de gerenciamento de ambiente de testes, o presente trabalho pretende possibilitar tais tarefas, através da estrutura de conhecimento criada.

Espera-se que pela estrutura de base de conhecimento criada, seja possível descobrir conhecimento novo e analisar os benefícios e as oportunidades de melhoria dos processos de teste de software. Poderão ser realizadas tarefas, como minerações, visualizações informativas, pesquisas e análises.

Os cinco itens acima podem ser resumidos na Figura 4.3.

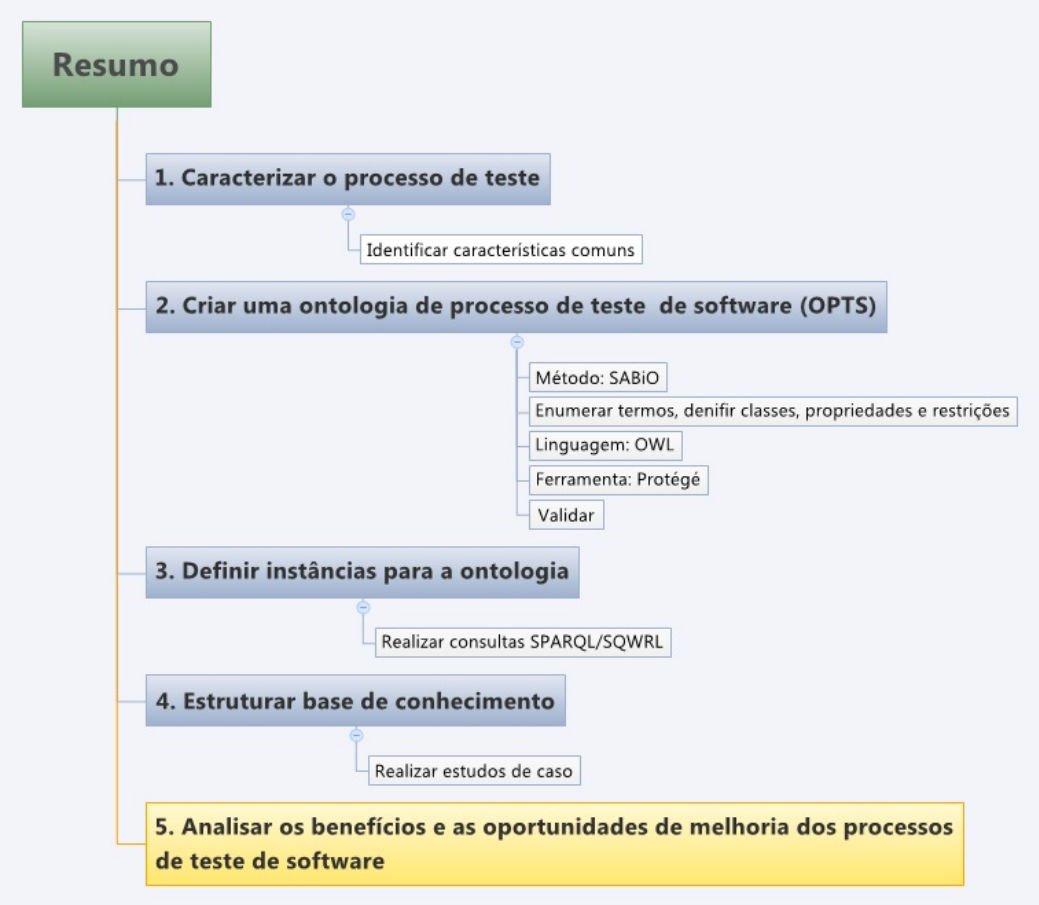

Figura 4.3 - Resumo das etapas a serem seguidas.

4.2 Validação dos Resultados

Além de ser complexo o desenvolvimento de uma ontologia, seja qual for o domínio, para desenvolver a ontologia OPTS será necessário o apoio de diversos especialistas na área de teste de software. Os especialistas poderão ajudar a validar e ter um consenso sobre o vocabulário e as restrições a serem consideradas.

A validação da ontologia também será realizada através da identificação dos conceitos, relações e axiomas necessários para responder a cada questão de competência estabelecida, visando a um compromisso ontológico mínimo com o domínio.

Já a validação da estrutura do modelo de descoberta de conhecimento em ambiente de teste de software, se dará por meio da realização de estudos de caso. Serão analisados os benefícios e as oportunidades de melhoria para a estratégia proposta em

estudos de caso no âmbito espacial.

4.3 Principais Contribuições

A contribuição geral deste trabalho está na criação uma estratégia de melhoria de processos de teste de software com base nos princípios da GC. A contribuição geral se desmembra em contribuições específicas, tais como: no levantamento de um vocabulário comum e restrições em um processo de teste de software; na definição de uma ontologia a fim de prover a conceituação e o conhecimento relacionados ao domínio de processos de teste; e na criação de uma estrutura de base de conhecimento considerado pela estratégia proposta (Figura 4.1).

O principal benefício da base de conhecimento será a possibilidade de se adotar uma estratégia mais produtiva para a aquisição de conhecimento em ambientes de teste de software, que permitirá reutilizar e compartilhar o conhecimento capturado. Acredita-se que a utilização da estratégia proposta também auxilie as organizações na preparação e realização da melhoria de seus processos de teste de software.

5 CRONOGRAMA DO TRABALHO

5.1 Atividades Propostas

As principais atividades realizadas e previstas para serem desempenhadas até a conclusão do Trabalho de Pesquisa em nível de doutorado são:

A. Obtenção dos créditos necessários de acordo com o regimento do programa: disciplinas, exame de língua (inglês) e estágio docência;

B. Levantamento bibliográfico, da literatura atual relacionada com o tema proposto para uma melhor familiarização e compreensão;

C. Preparação e defesa de qualificação;

D. Preparação e defesa da proposta;

E. Participação em conferências, eventos e congressos nacionais e/ou internacionais, que discutam a área de pesquisa proposta, com preparação e submissão de artigos para apresentar os resultados parciais da pesquisa utilizando para isso projetos reais como estudos de caso;

F. Análise crítica dos principais resultados obtidos;

G. Preparação do trabalho proposto seguindo as correções e alterações sugeridas na defesa da proposta;

H. Preparação de artigos para publicação dos resultados finais; e

I. Apresentação à banca (defesa final).

A Tabela 5.1 apresenta um cronograma relacionada com as principais atividades e datas a serem seguidas no programa de doutorado.

Tabela 5.1 - Cronograma

Etapas	2010				2011				2012	2013	Conc.
	3–5	6–8	9–11	12	1–5	6–9	8–10	11–12	1–12	1–12	
A											100%
B											70%
C											100%
D											50%
E											0%
F											0%
G											0%
H											0%
I											0%

REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. **Proc. ACM SIGMOD Int'l Conf. Management of Data**, v. 22, p. 207–216, 1993. 31

AMBRóSIO, A. M.; MATTIELLO, F.; CARDOSO, L. S.; SANTIAGO, V.; ARIAS, R.; VIJAYKUMAR, N. L.; LOUREIRO, G. **Experiências em projetos e uso de técnicas de verificação e validação de software em aplicações espaciais no INPE**. São José dos Campos, 2008. Acesso em Agosto de 2011. Disponível em: <http://mtc-m17.sid.inpe.br/col/sid.inpe.br/mtc-m17@80/2008/05.08.12.33/doc/publicacao.pdf>. 3

ANDERSIN, J. TPI - A Model for Test Process Improvement. **Seminar on Quality Models for Software Engineering, ACM Computing Classification System (CCS)**, Department of Computer Science, University of Helsinki, 2004. 2, 29

ANDRADE, M. T. T.; FERREIRA, C. V.; PEREIRA, H. B. B. Uma ontologia para a gestão do conhecimento no processo de desenvolvimento de produto. **Gest. Prod.**, Universidade Federal de São Carlos, São Carlos, v. 17, Nº 3, p. 537–551, 2010. 2, 7

BARBOSA, E. F. **Uma Contribuição ao Processo de Desenvolvimento e Modelagem de Módulos Educacionais**. Tese (Doutorado) — Tese de Doutorado em Ciências de Computação e Matemática Computacional., Universidade de São Paulo, USP, Brasil., 2004. 33, 37

BARBOSA, E. F.; NAKAGAWA, E. Y.; MALDONADO, J. C. Towards the establishment of an ontology of software testing. **In: 18th International Conference on Software Engineering and Knowledge Engineering (SEKE 2006)**, San Francisco, CA, p. 522–525, 2006. 33, 37

BARBOSA, E. F.; SOUZA, S. R. S.; MALDONADO, J. C. An experience on applying learning mechanisms for teaching inspection and software testing. **21st Conference on Software Engineering Education and Training (CSEE&T 2008)**, Charleston (SC), p. 189–196, 2008. 33

BARCELLOS, M. P.; FALBO, R. d. A.; ROCHA, A. R. A well-founded software process behavior ontology to support business goals monitoring in high maturity software organizations. **EDOCW '10 Proceedings of the 2010 14th IEEE International Enterprise Distributed Object Computing Conference Workshops**, p. 253–262, 2010. 19

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The goal question metric approach. 1994. Acesso em Maio de 2011. Disponível em: <http://www.cs.toronto.edu/~sme/CSC444F/handouts/GQM-paper.pdf>. 27

BASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. **Base de conhecimento em testes de software**. 2. ed. São Paulo: Martins Editora Livraria, 2007. 2, 22, 23, 25

BERTOLLO, G. **Definição de processos em um ambiente de desenvolvimento de software**. Dissertação apresentada ao Mestrado em Informática para obtenção do Grau de Mestre. Dissertação (Mestrado) — Universidade Federal do Espírito Santo (UFES), 2006. 18

BORGES, L. M. S.; FALBO, R. A. Managing software process knowledge. **Proceedings of the International Conference on CSITeA'2002**, Foz do Iguazu, Brazil, p. 227–232, 2002. 2, 17, 36

BURNSTEIN, I.; SUWANASSART, T.; CARLSO, R. Developing a testing maturity model for software test process evaluation and improvement. Washington, D.C., p. 581–589, 1996. 2, 29

CRAIG, R.; JASKIEL, S. P. **Systematic Software Testing**. Boston: Artech House Publishers, 2002. 29

CRESPO, A.; SILVA, O.; BORGES, C.; SALVIANO, C.; M., J. Uma metodologia para teste de software no contexto da melhoria de processo. **In: Simpósio Brasileiro de Qualidade de Software**, CenPRA - Centro de Pesquisa RenatoArcher, 2004. Acesso em Maio de 2011. 26

CUNHA, A. M. **Relatório Técnico do 5o Semestre do Projeto FINEP 5206/06.** São José dos Campos, 2010. 31

DAVENPORT, T. H.; PRUSAK, L. **Conhecimento Empresarial: como as organizações gerenciam o seu capital intelectual**. Rio de Janeiro: Campus, 1998. 8, 9

DAVIS, R.; SHROBE, H.; SZOLOVITS, P. What is a knowledge representation? **AI Magazine, Spring**, 1993. 11

DELAMARO, E.; MANDONADO, J. C.; M., J. **Introdução ao teste de software**. Rio de Janeiro: Ed. Elsevier, 2007. 22, 25

DEMING, E. **Out of the crisis**. MA: MIT Center for Advanced Engineering Study: MIT Press, 1986. 26

DESAI, A. Knowledge management and software testing. **International Conference and Workshop on Emerging Trends in Technology (ICWET 2011) - TCET**, Mumbai, Maharashtra, India, p. 767–770, 2011. 32, 33

DUARTE, K.; FALBO, R. Uma ontologia de qualidade de software. **VII Workshop de Qualidade de Software**, João Pessoa, Brasil, p. 275–285, 2000. 38

ERICSON, T.; SUBOTIC, A.; URSING, U. TIM - A Test Improvement Model. **In Proceedings of Softw. Test., Verif. Reliab.**, p. 229–246, 1997. 2, 29

FALBO, R.; BERTOLLO, G. Establishing a common vocabulary for software organizations understand software processes. **International Workshop on Vocabularies, Ontologies and Rules for the Enterprise**, Enschede, The Netherlands, 2005. 38

FALBO, R.; NATALI, A.; MIAN, P.; BERTOLLO, G.; RUY, F. Ode: Ontology-based software development environment. **IX Congreso Argentino de Ciencias de la Computación**, La Plata, Argentina, p. 1124–1135, 2003. 18, 38

FALBO, R. A. **Integração de conhecimento em um ambiente de desenvolvimento de software**. Tese (Doutorado) — Tese de Doutorado do programa de pós-graduação de engenharia da Universidade Federal do Rio de Janeiro (UFRJ,), no curso de Ciências em Engenharia de Sistemas e Computação., Rio de Janeiro, RJ - Brasil, 1998. 12, 17, 37, 38, 39

FALBO, R. A. Experiences in using a method for building domain ontologies. **In: International Workshop on Ontology in Action. Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering**, Banff, Canada, p. 474–477, 2004a. 15, 37

FALBO, R. A.; RUY, F. B.; BERTOLLO, G.; TOGNERI, D. F. Learning how to manage risks using organizational knowledge. **6th International Workshop on Learning Software Organization, LSO´2004F**, Banff, Canada, p. 7–18, 2004c. 38

FALBO, R. A.; RUY, F. B.; PEZZIN, J.; MORO, R. D. Ontologias e ambientes de desenvolvimento de software semânticos. **In: Actas de las IV Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento, JIISIC´2004**, Madrid, Espanha, p. 277–292, 2004b. 36

FAYYAD, U.; GREGORY, P.; P.SMYTH, P. Knowledge discovery and data mining: Towards a unifying framework. **Proceeding of the Second International Conference on Knowledge Discovery and Data Mining**, Portland, Oregon, 1996. 31

FENSEL, D.; HARMELEN, F. V. OIL: An Ontology Infraestructure for the Semantic Web. **IEEE Intelligent Systems**, p. 38–45, 2001. 14

FENSEL, D.; HARMELEN, F. v.; KLEIN, M.; AKKERMANS, H. On-to-knowledge: Ontology-based tools for knowledge management. **In Proceedings of the eBusiness and eWork 2000 (EMMSEC 2000) Conference**, 2000. 15

FERREIRA, E. **Uso de Mineração de Dados para Análise do Processo de Teste**. São José dos Campos - SP, 2010. Acesso em maio 2011. Disponível em: <http://urlib.net/8JMKD3MGP7W/38CDFNB>. 31, 32

FUGGETTA, A. Software process: A roadmap. **In Proc. of The Future of Software Engineering, ICSE'2000**, p. 25–34, 2000. 1

GREMBA, J.; MYERS, C. The IDEAL(SM) Model: A Practical Guide for Improvement. **In Software Engineering Institute (SEI) publication**, Bridge, v. 3, 1997. Acesso em Maio de 2011. Disponível em: <http://www.sei.cmu.edu/library/assets/idealmodel.pdf>. 1, 27

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing. **In Formal Ontology in Conceptual Analysis and Knowledge Representation**, Padova, Italy, 1993. 12

GU, X. L. G.; GUOCHANG, G.; YONGPO, L.; JI, W. Research and application of knowledge management model oriented software testing process. **Proceedings of the 11th Joint Conference on Information Sciences (2008)**, Shenzhen, China, 2008. 32

GUARINO, N. Formal ontology and information systems. **In Proceedings of International Conference in Formal Ontology and Information Systems - FOIS'98**, Trento, Italy, p. 3 – 15, 1998. 12, 13

HORROCKS, I. DAML+OIL: A Reason-able Web Ontology Language. **In: Proceedings of 8th International Conference on Extending Database Technology (EDBT)**, Prague, Czech Republic, p. 2–13, 2002. 14

IEEE. Institute of electric and electronic engineers. **Standard glossary of software engineering terminology**, Standard 610.12, 1990. 21, 24

ISO9000. **ABNT NBR ISO 9000 - Sistema de Gestão da Qualidade - Fundamentos e Vocabulário**. Associação Brasileira de Normas Técnicas (ABNT), 2006. Norma Brasileira. 21

KOOMEN, T.; POL, M. Test Process Improvement: A practical step-bystep guide to structured testing. **ACM Press, London, England**, 1999. 1, 26

LAMAS, E.; FERREIRA, E.; NASCIMENTO, M. R.; DIAS, L. A. V.; SILVEIRA, F. F. Organizational testing management maturity model for a software product line. **Seventh International Conference on Information Technology, IEEE Computer Society**, p. 1026–1031, 2010. 31

LAMAS, E. A. **Uma estrutura de maturidade operacional para gestão de teste de software aplicada a um projeto de monitoramento hidrológico**. Dissertação (Mestrado) — Dissertação de mestrado do curso de Engenharia Eletrônica e Computação. Área de Informática - Instituto Técnologico de Aeronáutica (ITA), São José dos Campos - SP, 2010. 3, 31

MAEDCHE, A.; VOLZ, R. The text-to-onto ontology extraction and maintenance environment to appear. **In: Proceedings of the ICDM Workshop on integratin data mining and knowledge management**, 2001. 13

MANTIS. 2011. Acesso em agosto de 2011. Disponível em: <http://www.mantisbt.org/>. 36

MARKKULA, M. Knowledge management in software engineering projects. **In: Proc. of the 11th International Conference on Software Engineering and Knowledge Engineering**, Kaiserslautern, Germany, p. 20–27, 1999. 2, 8

MARTIN, J. O.; AMAR, K. D. SQWRL: A Query Language for OWL. CEUR-WS.org, Stanford Center for Biomedical Informatics Research, v. 529, 2008. 39

MORO, R.; FALBO, R. A. Avaliação e melhoria de processos de software: Conceituação e definição de um processo para apoiar a sua automatização. **In: VIII Simpósio Brasileiro de Qualidade de Software**, Ouro Preto, p. 406–420, 2009. 2, 18

MPS.BR. **Melhoria de Processo do Software Brasileiro**. 2005. Acesso em abril 2011. Disponível em: <http://www.softex.br/mpsbr>. 28

MPT.BR. **Melhoria do Processo de Teste Brasileiro**. 2009. Acesso em março de 2011. Disponível em: <http://www.mpt.org.br/mptbr>. 2, 30

MYERS, G. J. **The art of software testing**. 2. ed. Canada: John Wiley and Sons, 2004. 22, 25

MYERS, P. S. **Knowledge Management and Organizational Design**. Butterworth-Heinemann: Reed Elsevier Group, 1996. 8

NAKAGAWA, E. Y.; BARBOSA, E. F.; MALDONADO, J. C. Exploring ontologies to support the establishment of reference architecture: An example on software testing. **In: 8th Working IEEE/IFIP Conference on Software Architecture (WICSA) / 3rd European Conference on Software Architecture (ECSA)**, Cambridge (UK), p. 249–252, 2009. 33

NARDI, J. C.; FALBO, R. A. Uma ontologia de requisitos de software. **In: IX Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software - IDEAS´2006**, 2006. 38

NATALI, A. C. C.; FALBO, R. Gerência de conhecimento de qualidade de software. **Proceedings of the 2nd Ibero-American Symposium on Software Engineering and Knowledge Engineering, JIISIC'2002**, Salvador - BA, 2002. 2

NONAKA, I.; TAKEUCHI, H. **Criação de conhecimento na empresa**. 1. ed. RJ: Campus, 1997. 9, 10

NOY, N. F.; MUSEN, M. A. **The knowledge model of Protégé-2000: combining interoperability and flexibility**. Stanford University, San Francisco, 2000. Acesso em março 2011. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.2129\&rep=rep1\&type=pdf>. 13

NUNES, V. **Integrando Gerência de Configuração de Software, Documentação e Gerência de Conhecimento em um Ambiente de Desenvolvimento de Software.** Dissertação (Mestrado) — Dissertação de Mestrado em Informática, Universidade Federal do Espirito Santo (UFES), Vitória, 2005. 38

O'LEARY, D. Using ai in knowledge management: knowledge bases and ontologies. **IEEE Intelligent Systems**, University of Southern California, v. 13, n. 3, p. 34–39, 1998b. 12

O'LEARY, D.; STUDER, R. Knowledge management: An interdisciplinary approach. **IEEE Intelligent Systems**, v. 16, No. 1, 2001. 7

O'LEARY, D. E. Enterprise knowledge management. **IEEE Computer Magazine**, p. 54–61, 1998a. 12

OWL. **OWL Web Ontology Language**. Word Wide Web Consortium (W3C), 2003. Acesso em abril de 2011. Disponível em: <http://www.w3.org/TR/2003/WD-owl-ref-20030331/>. 15

PARK, E. R.; GOETHERT, W. B.; FLORAC, A. **Goal-Driven Software Measurement - A Guidebook**. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996. 27

PAULK, M. Evolution of the sefs capability maturity model for software. **Software Process: Improvement and Practice, Pilot Issue, Spring**, v. 1, p. 3–15, 1995. 28

PAULK, M. C. Using the software CMM in small organizations. **In: Pacific Northwest Software Quality Conference, 16., with International Conference on Software Quality, Portland, Oregon**, v. 8, p. 350–361, 1998. 28

PECHEUR, C. Verification and validation of autonomy software at nasa. 2000. Acesso em agosto 2011. Disponível em: <http://ti.arc.nasa.gov/m/pub-archive/151h/0151%20(Pecheur).pdf>. 1

PRESSMAN, R. S. **Software engineering - a practitioner's approach**. 6. ed. New York: McGraw-Hill series in computer science, 2006. 1, 22

RDF. **Resource Description Framework (RDF)**. Word Wide Web Consortium (W3C), 2004. Acesso em abril de 2011. Disponível em: <http://www.w3.org/RDF/>. 14

RIOS, E. **Usando o MPS.BR para amadurecimento das equipes de teste de software**. [S.l.], 2009. Acesso em Abril de 2011. Disponível em: <http://www.iteste.com.br/LinkClick.aspx?fileticket=5PpKkE1ilCE%3D\&tabid=249\&mid=440>. 25

RIOS, E.; CRISTALLI, R. **Introdução ao TMM - Test Maturity Model**. [S.l.], 2007. Disponível em: <http://www.iteste.com.br/LinkClick.aspx?fileticket=qqVal9psbb4%3D\&tabid=249\&mid=440>. 28

RIOS, E.; MOREIRA, T. **Teste de software**. 2. ed. Rio de Janeiro: Alta Books, 2006. 23, 24

RIOS, J. A. Ontologias: alternativa para a representação do conhecimento explícito organizacional. **In Proceedings CINFORM - Encontro Nacional de Ciência da Informação VI**, Salvador-Bahia, 2005. 12

SANTIAGO, V.; VIJAYKUMAR, N. L.; GUIMARãES, D.; AMARAL, A. S.; FERREIRA, E. An environment for automated test case generation from statechart based and finite state machine-based behavioral models. In: FISRT IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION, 2003, Lillehammer, Norway. **Proceedings...** Lillehammer: IEEE Computer Society, 2008. p. 63–72. 3

SANTOS, G.; MONTONI, M.; FIGUEIREDO, S.; ROCHA, A. R. SPI-KM - Lessons Learned from Applying a Software Process Improvement Strategy Supported by Knowledge Management. **Lecture Notes in Computer Science**, p. 81–95, 2007. 2

SEI. **CMMI Capability Maturity Model Integration for Development (CMMI-DEV), Version 1.2**. Carnegie Mellon University. Pittsburgh, PA, 2006. Software Engineering Institute (SEI) . Acesso em março 2011. Disponível em: <http://www.sei.cmu.edu/reports/06tr008.pdf>. 1, 2, 28

SOFTEX. **MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral.** [S.l.], 2009. Acesso em Abril de 2011. Disponível em: <http://www.softex.br/mpsbr/_guias/guias/MPS.BR_Guia_Geral_2009.pdf>. 1

SPARQL. **SPARQL Query Language for RDF**. 2008. Acesso em setembro de 2011. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>. 39

SPENDER, J. C. Making knowledge the basis of a dynamic theory of the firm. **Strategic Management Journal**, v. 17, p. 45–62, 1996. Special issue. 8

STAAB, S.; MAEDCHE, A. Ontology engineering beyond the modeling of concepts and relations. **14th European Conference on Artificial Intelligence (ECAI). Workshop on Applications of Ontologies and Problem-Solving Methods**, Berlin, Germany, 2000. 13

SWARTOUT, B.; PATIL, R.; KNIGHT, K.; RUSS, T. Toward distributed use of large-scale ontologies. **In Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop**, Banff, Canada, 1996. 15

SWINKELS, R. **A comparison of TMM and other Test Process Improvement Models**. [S.l.], nov. 2000. Acesso em abril 2011. Disponível em: <http://www1.in.tum.de/static/lehrstuhl/files/teaching/ws0708/ManagementSoftwareTesting/12-4-1-FPdef.pdf>. 30

TESTLINK. 2011. Acesso em agosto de 2011. Disponível em: <http://www.teamst.org/>. 36

VALENTE, A. **Legal Knowledge Engineering - A Modelling Approach**. Inc. Boston, MA, USA: IOS Press, 1995. 39

W3C. **World Wide Web Consortium**. 1996. Extensible Markup Language (XML), Cambridge. Acesso em abril de 2011. 14

WU, Y. L. J.; XUEMEI, L. G. G. Investigation of knowledge management methods in software testing process. **International Conference on Information Technology and Computer Science**, v. 2, p. 90–94, 2009. 32, 33