Ministério da
Ciência, Tecnologia e Inovação

sid.inpe.br/mtc-m19/2014/01.21.12.55-TDI

KNOWLEDGE MANAGEMENT APPLIED TO SOFTWARE TESTING: AN ONTOLOGY BASED FRAMEWORK

Érica Ferreira de Souza

Doctorate Thesis Course Graduate in Applied Computing, guided by Drs. Nandamudi Lankalapalli Vijaykumar, and Ricardo de Almeida Falbo, approved in February 04, 2014.

URL of the original document:
<http://urlib.net/8JMKD3MGP7W/3FK67N8>

INPE
São José dos Campos
2014

PUBLISHED BY:

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) 3208-6923/6921
Fax: (012) 3208-6919
E-mail: pubtc@sid.inpe.br

Ministério da
Ciência, Tecnologia e Inovação

sid.inpe.br/mtc-m19/2014/01.21.12.55-TDI

KNOWLEDGE MANAGEMENT APPLIED TO SOFTWARE TESTING: AN ONTOLOGY BASED FRAMEWORK

Érica Ferreira de Souza

Doctorate Thesis Course Graduate in Applied Computing, guided by Drs. Nandamudi Lankalapalli Vijaykumar, and Ricardo de Almeida Falbo, approved in February 04, 2014.

URL of the original document:
<http://urlib.net/8JMKD3MGP7W/3FK67N8>

INPE
São José dos Campos
2014

Cataloging in Publication Data

Souza, Érica Ferreira.
So89k Knowledge management applied to software testing: an ontology based framework / Érica Ferreira de Souza. – São José dos Campos : INPE, 2014.
xxvi + 186 p. ; (sid.inpe.br/mtc-m19/2014/01.21.12.55-TDI)

Thesis (Doctorate in Applied Computing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2014.
Guiding : Drs. Nandamudi Lankalapalli Vijaykumar, and Ricardo de Almeida Falbo.

1. software testing 2. knowledge management 3. ontologies 4. knowledge management system. 5. knowledge reuse. I.Título.

CDU 005.94:004.9

Aprovado (a) pela Banca Examinadora em cumprimento ao requisito exigido para obtenção do Título de *Doutor(a)* em

Computação Aplicada

Dr. Rafael Duarte Coelho dos Santos

Presidente / INPE / SJCampos - SP

Dr. Nandamudi Lankalapalli Vijaykumar

Orientador(a) / INPE / SJCampos - SP

Dr. Ricardo de Almeida Falbo

Orientador(a) / UFES / Vitória - ES

Dr. Walter Abrahão dos Santos

Membro da Banca / INPE / São José dos Campos - SP

Dr. Monalessa Perini Barcellos

Convidado(a) / UFES / Vitória - ES

Dr. Otávio Augusto Lazzarini Lemos

Convidado(a) / UNIFESP / São José dos Campos - SP

Este trabalho foi aprovado por:

( ) maioria simples

(X) unanimidade

Aluno (a): *Érica Ferreira de Souza*

São José dos Campos, 04 de Fevereiro de 2014

In memory of my father Osvaldo Dias de Souza

ACKNOWLEDGEMENTS

First, I would like to gratefully thank my father Osvaldo Dias de Souza (in memorian), my mother Maria Aparecida Ferreira de Souza, my brothers Edevaldo Ferreira de Souza and Éder Luiz Ferreira de Souza, and my beloved Giovani Volnei Meinerz, for their support, encouragement, patience, and unconditional love and care during the time I devoted to this PhD thesis. It was crucial so I could have the strength to go ahead.

Next, I need to sincerely thank my advisors Dr. Nandamudi Lankalapalli Vijaykumar and Dr. Ricardo de Almeida Falbo, for their guidance, understanding, and most important, friendship during the development of this work. Without their mentorship, I may never have gotten to where I am today.

I must also express my gratitude to the Brazilian Institute for Space Research (INPE) for it valued institutional support and the necessary human and physical resources essential to perform all the research activities.

My sincere recognition and gratefulness to Coordination for the Improvement of Higher Education Personnel (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - CAPES) and São Paulo Research Foundation (Fundação de Amparo à Pesquisa - FAPESP), not only for providing the funding which allowed me to undertake this research, but also for giving me the opportunity to attend conferences and meet so many interesting worldwide researchers.

I also have to thank the members of my PhD committee, Professors Monalessa Perini Barcellos (UFES), Otávio Augusto Lazzarini Lemos (UNIFESP), Rafael Duarte Coelho dos Santos (INPE), Walter Abrahão dos Santos (INPE), for their input, valuable discussions, and helpful suggestions.

I would like to thank Dr. Adilson Marques da Cunha, from the Brazilian Aeronautics Institute of Technology (ITA), which was the coordinator of a Brazilian National Water Agency research and develepment project, whereby I got the real data applied on the case studies of my research.

I would like to acknowledge Dra. Gláucia Braga e Silva and Dr. Ronaldo Arias for the support and attention given to the case studies used in this thesis. I also would like to acknowledge Marcos da Silva Specimille for contribute in the final development of this thesis PhD.

Finally, I would like to thank all my fellow postgraduate students of the Applied Computing Division at INPE for helping me keep things in perspective and making everything much more pleasurable.

ABSTRACT

Software development organizations are seeking to add quality to their products. Testing processes are strategic elements to manage projects and product quality. However, advances in technology and the emergence of increasingly critical applications make testing a complex task and a large number of information is generated. In fact, software testing is a knowledge intensive processo In view of this, these organizations have shown a growing interest in knowledge management programs, which in turn support the improvement of testing procedures. In this context, testing knowledge should be captured and represented in an affordable and manageable way, and therefore, could make use of principles of Knowledge Management *(KM).* One of the main KM problems is how to represent knowledge. Ontologies are particularly important for KM. With respect to knowledge representation and Knowledge Management Systems (KMS), ontologies can help manipulating the knowledge items represented, by minimizing ambiguity and vagueness in the interpretation of the shared understanding regarding the domain. In this context, this work aims to define an ontology-based framework for guiding KM initiatives in the software testing domain, supported by KMS. This framework is subjected to a proof of concept, and as a result a Testing KM Portal was developed using actual knowledge items extracted from two software projects.

GESTÃO DE CONHECIMENTO APLICADO A TESTE DE SOFTWARE: UM FREMAWORK DE BASE ONTOLÓGICA

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. No entanto, os avanços tecnológicos e o surgimento de aplicações cada vez mais críticos tornam a atividade de teste uma tarefa complexa e um grande volume de conhecimento é gerado. De fato, o teste de software é um processo de conhecimento intensivo. Diante disso, tais organizações têm mostrado um crescente interesse por programas gerenciamento do conhecimento gerado, que consequentemente apoiam a melhoria dos processos de teste. Assim, o conhecimento de teste deve ser capturado e representado em uma forma acessível e controlável, e, portanto, pode fazer uso dos princípios da Gestão do Conhecimento *(GC)*. Um dos principais problemas na GC é a forma de representar o conhecimento. Ontologias são particularmente importantes para *GC*. No que diz respeito a representação do conhecimento e Sistemas de Gestão do Conhecimento (SGC), ontologias podem ajudar a manipular ontologias pode ajudar a manipular os itens de conhecimento representados, minimizando a ambiguidade e imprecisão na interpretação do entendimento comum sobre o domínio. Neste contexto, este trabalho tem define um framework de base ontológica para orientar iniciativas de GC no domínio de teste de software, apoiado por SGC. Este framework é submetido a uma prova de conceito, e como resultado, um Portal de GC para teste de software foi desenvolvido utilizando itens de conhecimento reais extraídos de dois projetos de software.

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

AOC Attitude and Orbit Control

DCTA *Departamento de Ciência e Tecnologia Aeroespacial*

DROPs Domain-related Ontology Patterns

E-OPL Enterprise Ontology Pattern Language

FDIR Fault Detection Isolation and Recovery

FSM Finite State Machines

HRPA Human Resource Participation

ICAMMH Amazon Integration and Cooperation for Modernization of Hydrological Monitoring

IT Information Technology

INPE *Instituto Nacional de Pesquisas Espaciais*

KDD Knowledge Discovery in Databases

KM Knowledge Management

KMS Knowledge Management Systems

LL Lesson Learned

MND-TMM Ministry of National Defense-Testing Maturity Model

MOAR Multi-Organization Arrangement

OBDH On-Board Data Handling

OCL Object Constraint Language

ODE Ontology-based software Development Environment

OP Ontology Patterns

OPL Ontology Pattern Language

OTD Organizational Teams

OWL Ontology Web Language

PAE Process and Activity Execution

PRPA Procedure Participation

PRT Procedure Taxonomy

RDF Resource Description Framework

RPA Resource Participation

ROoST Reference Ontology on Software Testing

SABiO Systematic Approach for Building Ontologies

SIA Inertial Systems for Aerospace Application

SLR Systematic Literature Review

SPARQL Protocol and RDF Query Language

SPO Software Process Ontology

SPP Software Process Planning

SPS Standard Process Structure

SP-OPL Software Process Ontology Pattern Language

SQL Structured Query Language

STOWS Software Testing Ontology for Web Service

T-KM Testing-Knowledge Management

TKMP Testing Knowledge Management Portal

TEAA Team Allocation

TEAR Team Roles

TOM Test Ontology Model

UFO Unified Foundational Ontology

UML Unified Modeling Language

XML eXtensible Markup Language

WEKA Waikato Environment for Knowledge Analysis

WPPA Work Product Participation

WPT Work Product Taxonomy

CONTENTS

Page

1 INTRODUCTION

During the last decades, with the emergence of new technologies, more advanced techniques have been applied in software development, in order to achieve high-quality software products (THRANE, 2011). Thus, more efficient techniques to qualify a software product should be incorporated in its development lifecycle, ensuring a well-managed process. This applies, in particular, in critical systems, such as systems for space applications, systems for nuclear power plants, medical equipment, among others.

The lack of use of mechanisms for assuring software quality, mainly in critical software, can cause significant losses. Classic examples in which defects in software were the main causes of failure are: the number of processors *Pentium Intel* due to a defect in math coprocessor (Floating Point Unit), becoming known as "*Pentium FDIV bug*" (PECHEUR, 2000); self-destruction of the rocket Ariane 5, due to an error in the control software (PECHEUR, 2000); Therac-25 medical electron accelerator (1985-1987)(LEVESON; TURNER, 1993), which caused several losses of human lives; and the Mars Climate Orbiter (1999) (NASA, 2014).

Quality is a desirable element for any kind of product including software. However, regardless of the product, obtaining quality is not an easy task. In the context of software quality, it is necessary to adopt a technical perspective and consider several factors that affect product construction and influence the judgment of users, such as: product complexity, conformance to requirements, number of people involved, tools, defect level, defect origins, and costs associated with the presence of defects and their removal (GODBOLE, 2006; KOSCIANSKI; SANTOS, 2007). Thus, it is clear that among these factors, Verification & Validation (V&V) performs an important role in assessing and achieving the quality of a software product.

Software development is an error prone process. To achieve software quality, it is essential to perform V&V activities throughout the software development process. Verification determines whether the developed products of a given activity conform to the requirements of that activity. Validation refers to whether the software satisfies its intended use and the user needs (IEEE, 1990).

V&V activities can be static and dynamic. Dynamic V&V activities require the execution of a program, while static V&V activities do not involve program execution. Static V&V are typically done by means of technical reviews and inspections. Dynamic V&V are done by means of testing (MATHUR, 2012). Thus, software testing

consists in dynamic V&V of the program behavior against the expected behavior (ABRAN et al., 2004).

Currently, software testing is considered as a process consisting of activities, techniques, resources and tools. Advances in technology and the emergence of increasingly critical applications make testing a complex task. During software testing, a large number of information is generated. In fact, software testing is a knowledge intensive process, and thus it is important to provide computerized support for tasks of acquiring, processing, analyzing and disseminating testing knowledge for reuse (ANDRADE et al., 2013). In this context, testing knowledge should be captured and represented in an affordable and manageable way, and therefore, software testing could make use of principles of KM.

With respect to KM, one of the main problems is how to represent knowledge. A KMS should support the integration of information from diversified sources, wherein a decision maker manipulates information that someone else has conceptualized and represented. So, a KMS must minimize ambiguity and imprecision in interpreting shared information. This can be achieved by representing the shared information using, for example, ontologies (KIM, 2000). An ontology is an explicit specification of a conceptualization (GRUBER, 1993). Ontologies are particularly important for KM, since they bind KM activities together, allowing a content-oriented view of KM (STAAB et al., 2001). An ontology can be used to define a shared vocabulary to be used in the KMS to facilitate knowledge communication, integration, search, storage and representation (BENJAMINS et al., 1998).

1.1 Motivation

Software Testing area has drawn a significant attention in both industrial and academic areas. Several different models for test process improvement have been used in order to guide testing. However, this alone is not enough to improve the organizational testing process. According to Andrade et al. (2013), one of the characteristics of software testing is that it has a large intellectual capital component and can thus benefit from the use of the experience gained from past projects.

Currently, software has become more and more critical with complex application domains, making the testing process increasingly important. Some examples are the systems developed at National Institute for Space Research (*Instituto Nacional de Pesquisas Espaciais* (INPE)) involving critical software embedded in scientific equipment on board satellites and/or stratospheric balloons. This makes the software

becomes increasingly more complex and, consequently, knowledge-intensive (NATALI et al., 2004).

In this context, the use of experiences of members or lessons learned must be taken into consideration. However, experiences gained are confined to each individual and this is not known or shared by other team members (at least not formally) (WU; XUEMEI, 2009; ABDULLAH et al., 2011; DESAI, 2011; ANDRADE et al., 2013). Furthermore, there is a loss of intellectual capital in the organization, due to limited knowledge of each individual and the turnover of staff (NOGESTE; WALKER, 2003; WU; XUEMEI, 2009; DESAI, 2011).

The adoption of principles of KM in software testing can assist specialists to promote the reuse of knowledge, to support processes of testing and even to encourage management decisions in organizations. There are many benefits of implementing KM in the software testing domain, such as (LIU et al., 2009; XU-XIANG; WEN-NING, 2010; LI; ZHANG, 2012; ANDRADE et al., 2013): (i) selection and application of better suited techniques; (ii) cost reduction; (iii) test effectiveness increase; and (iv) competitive advantages.

According to O'Leary (1998a), KM formally manages knowledge resources in order to facilitate access and reuse, typically by using advanced Information Technology (IT). IT-supported KM solutions are built around an organizational structure that integrates informal, semiformal, and formal knowledge to facilitate its access, sharing, and reuse (STAAB et al., 2001).

The theme addressed in this thesis, KM in software testing, has considered the relevance of projects from INPE and Departamento de Ciência e Tecnologia Aeroespacial (*Departamento de Ciência e Tecnologia Aeroespacial* (DCTA)). In INPE and DCTA, sectors related to construction of satellites and Satellite Launch Vehicle or other sectors as critical as these, have some kind of architecture or even automated environments to qualify software products considered critical (SANTIAGO JÚNIOR, V. A. and VIJAYKUMAR, N. L. and GUIMARÃES, D. and AMARAL, A. S. and Souza, E. F. , 2008; LAMAS, 2010a; LAMAS et al., 2010b; SANTIAGO JÚNIOR, V. A. and VIJAYKUMAR, N. L. and SOUZA, E. F. and GUIMARÃES, D. and COSTA, R. C., 2012). The use of strategies to guide on how to work properly with the knowledge within these organizations assist in supporting the software testing processes, thus ensuring a higher quality of the software product generated in these projects.

1.2 Problem Characterization

Several studies have reported on the problem of software testing knowledge reuse within organizations (WU; XUEMEI, 2009; ABDULLAH et al., 2011; LI; ZHANG, 2012; ANDRADE et al., 2013; JANJIC; ATKINSON, 2013). The major problem in organizations are low reuse rate of knowledge and barriers in knowledge transfer. This occurs because most of the knowledge in organizations is not treated and it becomes difficult to articulate (ABDULLAH et al., 2011; LI; ZHANG, 2012; ANDRADE et al., 2013).

The main issue is that test teams do not benefit from the knowledge and experience acquired, as these are limited to a single individual and, therefore, it becomes more difficult to bring this knowledge to the organizational level. As a consequence, the same mistakes are made over and over again and successful practices are not repeated. Other important issue is that employees are reluctant to share their knowledge as they feel that retaining this knowledge is an advantage over their colleagues and their superiors (KERKHOF et al., 2003; WU; XUEMEI, 2009; ABDULLAH et al., 2011; DESAI, 2011; ANDRADE et al., 2013).

All the knowledge and experience acquired during a software project can be reused in the future. Software testing, in general, can benefit from reusing modules, test cases, testing techniques, lessons learned and personal experiences. To enable testing knowledge reuse, software organizations should be able to capture this knowledge and make it available for their staff. However, there are only a few KM solutions in the context of software testing, as we could perceive by means of a systematic mapping presented in (SOUZA et al., 2013a).

Even when some KM strategy is applied, it is not always successful to achieve organizational learning, because the existing communication systems are not appropriate (WU; XUEMEI, 2009). There are many difficulties in implementing knowledge acquisition, coding, storage and searching functionalities effectively in a KMS, because it involves all the aforementioned problems, mainly time and interest of the employees (WU; XUEMEI, 2009; ANDRADE et al., 2013).

1.3 Objectives

The general objective of this thesis is to define a KM framework to manage software testing knowledge, so that different testing knowledge items are collected, shared, reused and improved throughout the organization. This general objective can be divided into the following specific objectives:

1. Identify the state of the art on KM applied to software testing through a Mapping Study;

2. Conduct a Systematic Literature Review (SLR) with the purpose of investigating how widespread are the use of ontologies for managing software testing, and the characteristics of such ontologies.

3. Develop a reference ontology on software testing to establish a common conceptualization of the software testing domain.

4. Develop a process defining a set of directions for implementing of KM in Software Testing; and

5. Employ the proposed framework to build a KMS for managing software testing knowledge, as a proof of concept of the feasibility of applying the proposed framework.

1.4 Research Method

This work started with a systematic mapping with the goal of making evident some aspects associated to the employment of KM in software testing and research efforts that can drive future research (KITCHENHAM et al., 2011). This mapping, which is presented in Chapter 2, achieved the following conclusions (SOUZA et al., 2013a): (i) the major problem in software organizations related to software testing are low knowledge reuse rate and barriers in knowledge transfer; (ii) reuse of test cases is a perspective that has received more attention; (iii) there is a great concern with both explicit and tacit knowledge.

KM applied to the software testing domain has shown to be a very promising approach, since KM techniques can help handling knowledge within an organization in several respects, as shown by the systematic mapping. On the other hand, the systematic mapping also showed that KM in software testing still seems to be a challenge. The findings of the mapping also motivated to advance in this research line.

The KM community recognizes ontologies as an important instrument for representing KM (BENJAMINS et al., 1998; MAEDCHE; VOLZ, 2001; AHMAD et al., 2011; VALASKI et al., 2012). Nevertheless, few studies have actually used an ontology-based approach for KM in the software testing domain. Thus, investing efforts in such research area seems to be promising. Looking for a domain ontology that could

be used in a KM initiative in software testing, a SLR was conducted to investigate existing ontologies in the software testing domain (SOUZA et al., 2013c). This SLR is presented in Chapter 3. 12 ontologies addressing this domain were identified. To analyze these ontologies, some of the characteristics pointed out by D'Aquin and Gangemi (2011) were considered as characteristics that are presented in "beautiful ontologies", namely: (i) having a good domain coverage; (ii) implementing an international standard; (iii) being formally rigorous; (iv) implementing also non-taxonomic relations; (v) following an evaluation method; and (vi) reusing foundational ontologies.

The main findings obtained from this SLR are: most ontologies have limited coverage; the studies do not discuss how the ontologies were evaluated; none of the analyzed testing ontologies is truly a reference ontology; and none of them is grounded in a foundational ontology. In a nutshell, the software testing community should invest more efforts to get a well-established reference software testing ontology. The investigated ontologies were inappropriate for the purposes of this thesis. In order to properly manage software testing knowledge, a software testing ontology is necessary. More specifically, a reference domain ontology, i.e. a domain ontology that is constructed with the main goal of making the best possible description of the domain as realistic as possible. A reference domain ontology is a special kind of conceptual model representing a model of consensus within a community. It is a solution-independent specification with the aim of making a clear and precise description of domain entities for the purposes of communication, learning and problem-solving (GUIZZARDI, 2007). A reference ontology on the software testing domain can be used for several KM-related purposes, such as for structuring knowledge repositories, for annotating knowledge items, and for making searching easier. Thus, building an ontology considering the aspects analyzed in the SLR became one of goals of this thesis, and a Reference Ontology on Software Testing (ROoST) (SOUZA et al., 2013b) was developed, which is presented in Chapter 4.

In order to develop ROoST, ontology patterns from a software process pattern language, SP-OPL (FALBO et al., 2013), were reused. SP-OPL is a core ontology on software processes. As a core ontology, SP-OPL provides a precise definition of the structural knowledge in the field of software processes and spans across different application domains, for example, in software testing (FALBO et al., 2013). In addition, patterns from Enterprise Ontology Pattern Language (E-OPL) (FALBO et al., 2014) were also used to address aspects such as Organization Arrangement, Definition Team, Institutional Roles, Institutional Goals, and Human Resource Management.

It is worthwhile to print out that both, SP-OPL and E-OPL are grounded on the Unified Foundational Ontology (UFO) (GUIZZARDI, 2005; GUIZZARDI et al., 2008).

ROoST has been developed in a modular way. Currently, ROoST covers aspects related to *Software Testing Process* and its *Activities*, *Artifacts* that are used and produced by those activities, *Testing Techniques* for test case design, and the *Software Testing Environment*, including hardware, software and human resources. Finally, in order to evaluate ROoST, specific V&V activities for ontologies were performed. ROoST evaluation started with a verification activity, where the defined concepts, relations and axioms are able to answer the competency questions. To validate ROoST, its concepts and relations were instantiated with individuals extracted from an actual project, in order to check whether the ontology was able to represent concrete situations of the real world.

The systematic mapping (SOUZA et al., 2013a) showed that one of the most important research challenges in KM applied to software testing is how effectively one can integrate KM with software testing so that knowledge items can be shared and reused. Clearly, ROoST helps to address some of these issues, but it is not enough. A process is also necessary to provide directions on how to identify KM goals that are specific for a particular software testing organization, and how to achieve a KMS to support the organization's KM initiative in software testing. Thus, the next step was to define a process for this purpose, which is presented in Chapter 4. Once defined this process, a framework for guiding testing organizations in the accomplishment of KM initiatives in software testing has been developed, called T-KM framework. It is worthwhile to point out that, in this thesis, the term framework was used as defined by Wong and Aspinwall (2004), i.e., to designate a set of basic assumptions or fundamental principles that forms the underlying basis for action. In other words, a framework can be interpreted as a structure that comprises relevant entities or a set of guiding principles and ideas that support a discipline. Thus, the T-KM framework consists of two main components: ROoST and a process to develop a KMS to support managing software testing knowledge.

As a proof of concept, T-KM framework was applied in a general scenario, since there was no available testing organization to serve as a case study. As the mapping results showed, managing testing knowledge is not an easy task. So, firstly, it is necessary to identify essential knowledge items of a sub-topic of the software testing domain to be dealt with. Thus, in order to identify such elements, a survey was performed. The aim of the survey was to define a scenario to apply KM in software testing.

Nine questions were defined, and the questionnaire was sent to testing experts. 86 experts participated in the survey. The questions in the survey are related to the mapping, as well as to the conceptualization described by ROoST. Considering the survey results, and based on ROoST, a Testing KMS was developed. The KMS main functionalities are: knowledge items creation and, evaluation, as well as search, retrieval and valuation of these items. The system was evaluated by project leaders in two real scenarios. The application of T-KM framework is discussed in Chapter 5.

The thesis document was prepared as the work progressed, documenting the results of the studies, the techniques that were employed and the solutions adopted. Also, during this work, scientific papers were written and submitted/published in the following vehicles:

1. **Érica F. de Souza**; Leando Evaristo; Ricardo A. Falbo; Nandamudi L. Vijaykumar. "*Using ontologies to build a database to obtain strategic information in decision making*". V Seminário de Pesquisa em Ontologias do Brasil - VII International Workshop on Metamodels, Ontologies, Semantic Technologies, ONTOBRAS - MOST. Recife/PE, 2012.

2. **Érica F. de Souza**; Ricardo A. Falbo; Nandamudi L. Vijaykumar. "*Knowledge Management Applied to Software Testing: A Systematic Mapping*". The 25th International Conference on Software Engineering and Knowledge Engineering (SEKE 2013). Boston/USA, 2013.

3. **Érica F. de Souza**; Ricardo A. Falbo; Nandamudi L. Vijaykumar. "*Ontologies in Software Testing: A Systematic Literature Review*". VI Seminário de Pesquisa em Ontologias do Brasil - ONTOBRAS 2013. Belo Horizonte/MG, 2013.

4. **Érica F. de Souza**; Ricardo A. Falbo; Nandamudi L. Vijaykumar. "*Using Ontology Patterns for Building a Reference Sofware Testing Ontology*". The 8th International Workshop on Vocabularies, Ontologies and Rules for the Enterprise and Beyond (VORTE 2013). Vancouver/Canada, 2013.

1.5 Organization of this Thesis

This introductory chapter presented the context, the motivation that led to the development of this thesis, as well as the objectives of this work and the research method followed. The remainder of this thesis is organized as follows:

- *Chapter 2 - KNOWLEDGE MANAGEMENT IN SOFTWARE TESTING:* this chapter presents the main concepts in the research areas studied in this thesis, namely KM and software testing. A systematic mapping on KM in software testing is also presented.

- *Chapter 3 - ONTOLOGIES FOR SOFTWARE TESTING:* this chapter briefly discuss ontologies, and related concepts, as well as presents the systematic literature review on software testing ontologies.

- *Chapter 4 - AN ONTOLOGY-BASED FRAMEWORK FOR KNOWLEDGE MANAGEMENT IN SOFTWARE TESTING:* this chapter presents the main contribution of this thesis: the ontology-based framework for KM in software testing, called *T-KM*.

- *Chapter 5 - APPLICATION OF THE PROPOSED FRAMEWORK:* this chapter discusses the application of the proposed framework as a proof of concept.

- *Chapter 6 - FINAL CONSIDERATIONS:* this chapter presents the conclusions of this work, its main contributions, as well as future directions to follow.

- *ANNEX A - ONTOLOGY PATTERN LANGUAGE: SP-OPL and E-OPL:* presents the SP-OPL and E-OPL patterns used in this thesis to develop ROoST.

- *APPENDIX A - SURVEY: KM IN SOFTWARE TESTING:* presents details of the survey and its results, which were used to define a scenario to apply KM in software testing.

- *APPENDIX B - LOADING EXISTING KNOWLEDGE ITEMS:* presents the procedure followed to load existing knowledge items in the scenario studied.

2 KNOWLEDGE MANAGEMENT IN SOFTWARE TESTING

In this chapter, some of the most important concepts in the research areas studied are discussed briefly. The main concepts of Software Testing are described in Section 2.1. Section 2.2 presents an overview about Knowledge Management (KM). Finally, Section 2.3 presents aspects associated with applying KM in software testing through a Systematic Mapping Study.

2.1 Software Testing

Software testing is a set of activities with the main objective to contribute to the quality of the products generated. According to IEEE (1990), quality is the degree to which a set of characteristics satisfies the requirements. In other words, it can be said that if any product or service meets the specified requirements, the same product or service has, in principle, the desired quality.

To achieve quality software products, it is essential to perform Verification & Validation (V&V) activities throughout the software development process. The *IEEE Standard Glossary of Software Engineering Terminology* (IEEE, 1990) makes the following definitions:

> **Verification**: is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the beginning of that phase.
>
> **Validation**: is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

Three important concepts related to V&V are: *fault*, *error* and *failure*. Considering the definition established by *IEEE Standard Glossary of Software Engineering Terminology* (IEEE, 1990), the concepts are defined as follows:

> **Fault**: is an incorrect step, process, or data definition in a computer program.
>
> **Error**: the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.
>
> **Failure**: the inability of a system or component to perform its required functions within specified performance requirements.

V&V activities can be static and dynamic. Dynamic V&V activities require the execution of a program, while static V&V activities do not. Static V&V are typically done by means of technical reviews and inspections. Dynamic V&V are done by means of testing (MATHUR, 2012). Thus, software testing consists of the dynamic V&V of the behavior of a program on a finite set of test cases, against the expected behavior (ABRAN et al., 2004). A test case is a set of inputs, execution conditions and a expected result of a certain program or unit. The simplest definition of software testing is "Testing is the process of executing a program with the intent of finding errors" (MYERS, 2004). In this sense, a testing activity can be considered successful when the program under test fails (PRESSMAN, 2006).

Until the 90s, tests were conducted by the software developers and this was considered an inappropriate methodology. Information extracted from these tests did not allow the detection of all the defects. The situation became worse when applications started becoming more complex due to the emergence of new technologies, driving organizations to seek new solutions to improve software quality. It was from this decade that software test began to be treated not as an activity of the development process, but as an independent process (BASTOS et al., 2007). The main objective of the testing process is to minimize the risks caused by defects from the development process and add quality to the product generated (BASTOS et al., 2007).

Testing activities are supported by a well defined and controlled testing process. The testing process consists of several activities, namely: *Test Planning, Test Case Design, Test Execution* and *Test Result Analysis* (ABRAN et al., 2004; BASTOS et al., 2007; BLACK; MITCHELL, 2008). Briefly, key aspects of *Test Planning* include, among others, coordination of personnel, management of available test facilities and equipment, scheduling testing activities, and planning for possible undesirable outcomes. *Test Case Design* aims at designing the test cases to be run. Test Cases should be implemented as Test Scripts. During *Test Execution*, test cases are run, producing actual results. Finally, in the *Test Result Analysis*, test results are evaluated to determine whether or not tests have been successful in identifying defects.

In addition to the key concepts and software testing activities, techniques, levels, artifacts and software testing environment (including hardware, software and human resources) are also integrated into the testing process and are described briefly below.

i) **Testing Techniques**

The only way to ensure the correctness of software would be through an exhaustive

test, executing the software with all combinations of input values. However, this practice is not feasible, because the input domain can be infinite or at least very large (MYERS, 2004). Therefore, techniques to select a subset of test data should be used. The principle underlying such techniques is to be as systematic as possible to identify a representative set of program behaviors, for instance, considering subclasses of the input domain, scenarios, states, and data flows (ABRAN et al., 2004). Generally, test technique classification is based on how test cases are generated (MATHUR, 2012).

There are a variety of ways to generate tests from different types of techniques, some are listed in Table 2.1. Testing techniques types can be classified, among others, according to Mathur (2012), into: *Black-box Testing*, which generate test cases relying only on the input/output behavior, without the aid of the code that is under test. When the requirements are informally specified, one could use ad hoc techniques or heuristics such as equivalence partitioning and boundary-value analysis to generate test; *White-box Testing* refers to the activity wherein code is used in the generation of or the assessment of test cases. Control flow, Data flow and Coverage testing can be used for direct as well as indirect code-based test generation; *Defect-based Testing*, which are based on devising test cases specifically aimed at revealing categories of likely or predefined faults such as Mutation Testing; *Model-based Testing* occurs when the requirements are formally specified, for example, using one or more mathematical or graphical notations such as Statecharts, Finite State Machines (FSM) and others.

Table 2.1 - Classification of techniques types (MATHUR, 2012)

Artifact	**Tecnique**	**Example**
Requirements (informal)	Black-box	Ad hoc Testing, Boundary-value analysis, Category partition, Classifications trees, Cause-effect graphs, Equivalence partitioning, Random testing
Code	White-box	Coverage testing, Data-flow testing, Structural testing, Control-flow testing, Test minimization
Formal model: Graphical or mathematical specification	Model-based specification	Statechart testing, FSM testing

Continues

Table 2.1 - Conclusion

Artifact	**Tecnique**	**Example**
Code	Defect-based Testing	Mutation testing

ii) **Test Level**

Another important feature of testing is that test is often categorized based on the phase in which it occurs, that is, test usually is performed at different levels. Three important test levels can be distinguished, namely: *Unit Testing*, *Integration Testing* and *System Testing*.

In *Unit Testing*, the focus is on the unit or the individual components that have been developed. The goal is to ensure that the unit functions correctly in isolation. When units are integrated and a large component or a subsystem formed, programmers do *Integration Testing* of the system. In *Integration Testing*, the goal is to ensure that a collection of components function as desired. On the other hand, when the entire system has been built, its testing is referred to as *System Testing* (ABRAN et al., 2004; MATHUR, 2012).

iii) **Test Artifact**

Test artifacts are produced and used throughout the testing process. Documentation is an integral part of the formalization of the test process (ABRAN et al., 2004). According to IEEE (1998), test artifact may include, among others, Test Plan, Test Procedure, Test Case, Test Results.

During the planning activity the key test artifact is developed: *Test Plan*. The *Test Plan* describes how the test should be performed and provide avenues for future activities. In test case design activity the *Test Case* artifacts are generated. *Test Case* artifact defines test cases, which includes the input data, expected results, steps and general conditions for the exercise test case. During test execution activity, test cases are run which should also be documented. The results are recorded in *Test Results* artifact. Finally, during a test result analysis activity, *Test Results* are analyzed and a *Test Analysis Report* artifact is produced (BASTOS et al., 2007).

iv) **Test Environment**

According to Perry (2006), software testers are most effective when they work in an environment that encourages and supports well-established testing policies and procedures. The goal of a testing environment is to cause the software under test to exhibit true production behavior while being observed and measured outside of its production environment (EVERETT; RAYMOND, 2007).

The test environment is any structure where the test is performed. The elements considered in this structure, among others, are: hardware, software and human resources. The human resource or test team is responsible for setting up an isolated test environment, organized, representative and measurable to ensure the discovery of incidents, and most importantly, provide assurance that there was no external influence. It is necessary for the preparation of the environment is performed as soon as possible, and your basic needs (hardware and software) must be identified in the initial stage of the project (BASTOS et al., 2007).

2.2 Knowledge Management

KM is strongly related to the definition of what is knowledge. However, knowledge can be confused with data and information, but there is a difference. According to Davenport and Prusak (2000), data is a set of discrete, objective facts about events. In an organizational context, data is usually described as structured records of transactions. Modern organizations usually store data in some sort of technology system. The information is a message, usually in the form of a document. Possessing a sender and a receiver, the information is intended to change the way the recipient sees something, exerting an impact on one´s judgement and behavior. Different from data, information has meaning, that is, it has relevance and purpose, and is organized for any purpose. Knowledge derives from information as information derives from data. If information is to become knowledge, humans must do virtually this work.

Knowledge is one of the organization's most valuable assets. According to Nonaka and Takeuchi (1997), there are two main types of knowledge: tacit and explicit. Nonaka and Takeuchi (1997) use the tacit-explicit distinction to differentiate unarticulated and articulated stocks of knowledge. Tacit knowledge is the subjective and experience-based knowledge, that cannot be documented, and typically remains only in people's mind. This type of knowledge depends on personal experience and involves intangible factors such as beliefs, perspectives, values and intuition (NONAKA; TAKEUCHI, 1997). Tacit knowledge covers knowledge that is unarticulated and tied

to the senses, movement skills, physical experiences, intuition, or implicit rules of thumb. Even if we try hard, this type of knowledge cannot fully be articulated (NONAKA; KROGH, 2009). Explicit knowledge, in turn, represents the objective and rational knowledge that can be documented, and thus it can be accessed by multiple people (NONAKA; TAKEUCHI, 1997). Explicit knowledge can be uttered and captured in drawings and writing, and can be easily used and shared. The concept of "knowledge conversion" explains how tacit and explicit knowledge interact along a continuum (NONAKA; KROGH, 2009).

Nonaka and Takeuchi (1997) considers the creation of knowledge as a continuous and dynamic interaction between tacit and explicit knowledge, as shown in Figure 2.1, held by four different modes of knowledge conversion, namely:

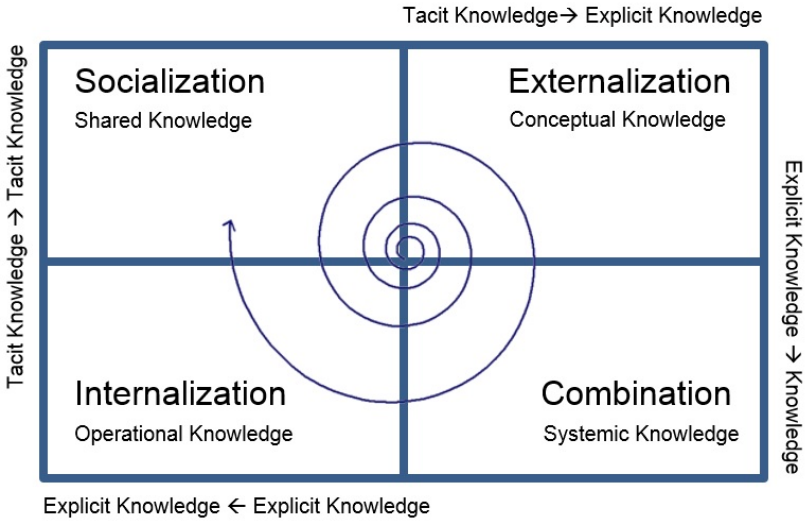

Figure 2.1 - Knowledge Spiral
SOURCE: Adapted from (NONAKA; TAKEUCHI, 1997)

- **Socialization:** It is the transmission of tacit knowledge from one individual to another. Socialization occurs usually through dialogue, sharing of experiences, brainstorming sections (formal meetings in order to generate solutions to specific problems), among others. The tacit knowledge held by individuals is the basis for the creation of organizational knowledge;
- **Externalization:** Is the transformation of tacit knowledge into explicit knowledge through the symbolic representation of tacit knowledge. This

usually occurs through the use of models, concepts, hypotheses, descriptions, drawings;

- **Combination:** It is the combination of different sets of explicit knowledge to generate new explicit knowledge. For example, the combination of documents may generate new documents; and
- **Internalization:** The process of incorporation of explicit knowledge to tacit knowledge. Typically, the internalization occurs through reading documents, observation of practice, conducting activities, that is, from existing explicit knowledge, new tacit knowledge are generated.

Finally, KM can be defined as a set of organizational activities that must be performed in a systematic manner with the purpose of acquiring, organizing and communicating both tacit and explicit knowledge in the organization, so that other members can use this knowledge to make their work more effective and productive. The main goal of KM is to make organizational knowledge accessible, reusable, promote knowledge storage and sharing, as well as the emergence of new knowledge (O'LEARY; STUDER, 2001). Davenport and Prusak (2000) has defined KM as a method that simplifies the process of sharing, distributing, creating, capturing and understanding of a company's knowledge. Bukowitz and Williams (1999) concludes that KM is the process by which the organization generates wealth from its knowledge or intellectual capital. In a simple form, KM is the process through which organizations generate value from their intellectual and knowledge-based assets. Most often, generating value from such assets involves sharing them among employees, departments and even with other companies in an effort to devise best practices.

According to Zack and Serino (2000), KM can be viewed as the development and leveraging of organizational knowledge to increase organization's value. Organizational knowledge creation aims at making available and amplifying knowledge created by individuals as well as crystallizing and connecting it to an organization's knowledge system (NONAKA; KROGH, 2009). KM entails formally managing knowledge resources in order to facilitate access and reuse of knowledge, typically by using advanced information technology. A wide range of technologies have been used in the development of KM systems, such as databases, data mining, intranets and internet, intelligent information retrieval, intelligent agents, case-based reasoning, yellow pages, ontologies, visualization models, groupware and so on (O'LEARY, 1998a; LIAO, 2003).

Ontologies, in special, are important for KM (BENJAMINS et al., 1998; MAEDCHE et al., 2003; AHMAD et al., 2011; VALASKI et al., 2012). They constitute the glue that binds KM activities together, allowing a content-oriented view of KM (STAAB et al., 2001). Ontologies define the shared vocabulary used in the KM system to facilitate communication, integration, search, storage and representation of knowledge (O'LEARY, 1998a).

Independently of the technology adopted, it is important that a set of basic activities should be considered to manage knowledge systematically. These activities are part of the KM process. KM processes are the sequential steps of conducting KM in an organization. According to Staab et al. (2001), once a KM system is fully implemented in an organization, knowledge processes are essentially used. It is possible to find several published literature that proposes processes to be used for the systematic KM within organizations (HENDRIKS; VRIENS, 1999; PROBST et al., 2000; DAVENPORT; PRUSAK, 2000). Some of the main activities are:

i) Knowledge Item Creation

Knowledge needs to be created or converted such that they it the conventions of the company, e.g. to the knowledge management infrastructure of the organization (STAAB et al., 2001). However, a great challenge for software organizations is to know what information is really useful among all the information generated within the organization. The information must be digested and organized to become meaningful and useful so that it can help in future decision-making knowledge. Thus, it is first necessary to define what types of knowledge items are relevant to the needs of the organization itself (NUNAMAKER et al., 2001; COELHO, 2010).

It is possible to find in the literature several types of knowledge items that are typically considered by software organizations (MONTONI, 2003; NATALI, 2003). Some are: process descriptions, new technologies, lessons learned, best practices, ideas and knowledge about the application domain. According to O'Leary (1998a) types of knowledge items usually include manuals, charts, news, client information, competitor information and knowledge gleaned from work processes.

ii) Knowledge Item Evaluation

Not every knowledge item created by a member must be available to the organization. Some knowledge items may have some inconsistencies, such as: organization

members can generate knowledge items in unsuitable or even not understandable or poorly explained formats; items of little use to the organization, and redundant items. Thus, it is important to evaluate an item of knowledge before it is available to the entire organization, particularly its relevance and verifying its correctness. According to Winch (1999), perceptions and interpretations of knowledge vary from person to person and, therefore, the knowledge must be filtered and evaluated before being made available.

According to Coelho (2010), allowing knowledge items to be evaluated by experts, without any criteria, can generate problems. The ideal way is to structure the evaluation of knowledge items, to prevent these problems from occurring. Each organization can define, according to their needs, the criteria to be used for evaluation of knowledge items. Montoni (2003) suggests the use of some criteria such as *Correction, Completeness, Consistency, Utility* and *Applicability.*

iii) Knowledge Item Dissemination

Mechanisms which allow search and retrieval of knowledge items can be developed to assist in the knowledge dissemination and to ensure that relevant information is made available to interested parties at the appropriate time to perform their tasks (COELHO, 2010). It is important to ensure that interested people can actually use the knowledge items available.

Knowledge repositories may become considerably large. Thus, the search for information in these repositories is an extremely critical task and must be done efficiently (O'LEARY, 1998a). Moreover, the dissemination of knowledge items is strongly related to indexing knowledge items in the repository. When entering knowledge items in the repository it is important to define appropriate classification schemes to facilitate the retrieval of knowledge items when necessary. Besides indexing/classification scheme, it is also important to define how and when knowledge items can be retrieved and presented to organization members (COELHO, 2010).

iv) Knowledge Item Valuation

The valuation has focused on enabling mechanisms to evaluate available knowledge items, aiming to capture feedback from your users. Ruggles (1998) highlights the importance of measuring the value of knowledge assets available in the organization, indicating that it is important to establish a mechanism for evaluating the utility

of knowledge assets from the point of view of the users. Thus, it becomes possible to collect data for measures, such as utilization rate and application of assets, and enables the discovery of assets with greater value to the organization.

v) Knowledge Item Maintenance

The maintenance of knowledge items is essential to ensure the reliability of stored knowledge and to enable KM goals are achieved properly (COELHO, 2010). For this, well-defined criteria, so that relevant knowledge can evolve, should be established, and knowledge without relevance can be deleted. Maintaining knowledge items can be taken based on data collected during the item valuation (user feedback) (NATALI, 2003). These data can provide useful information to identify knowledge items that need to be updated or removed. According to Abecker et al. (1998), the lack of maintenance of knowledge items is one of the main reasons why a Knowledge Management System (KMS) does not get the desired success in organizations.

vi) Knowledge Discovery

In a KMS, at every moment data are being stored, forming large volumes of data at a dramatic pace. The stored data can contain hidden useful information (knowledge) of great relevance to the business. A mining on these data can be performed. Data mining is the application of specific algorithms for extracting patterns from data. Data mining integrates the Knowledge Discovery in Databases (Knowledge Discovery in Databases (KDD)), process knowledge data structuring. The basic flow of KDD process steps is illustrated in Figure 2.2. Most work in the literature on KDD has focused on data mining. However, the other steps are as important (and probably more so) for the successful application of KDD in practice. Data mining is a step in the KDD process that consists of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns (or models) over the data (FAYYAD et al., 1996b).

Data mining methods are used in the identification of relevant information in large volumes of data, such as Classification, Regression, Clustering, Summarization, Association Rule, Dependency Modeling, among others (FAYYAD et al., 1996b).

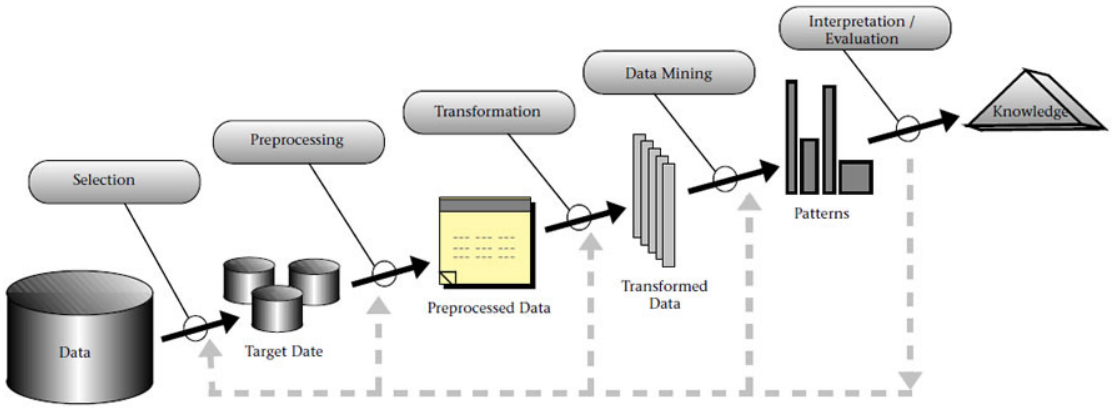

Figure 2.2 - Overview of the steps that compose the KDD Process
SOURCE: (FAYYAD et al., 1996b)

2.3 Knowledge Management applied to Software Testing

In the context of software testing, KM can be used to capture knowledge and experience generated during the testing process. However, this knowledge is usually stored on paper or in people's minds. When a problem arises, the team members look for experts in their own work environment, relying on people they know, or look for documents. Unfortunately, paper has limited accessibility and it is difficult to update (O'LEARY, 1998a). On the other hand, in a large organization, it can be difficult to locate who knows a certain matter, and knowledge in people's minds (tacit knowledge) is lost when individuals leave the organization. Therefore, software testing knowledge has to be systematically collected, stored in an organizational repository, and shared across the organization. In other words, KM is necessary.

This section presents an investigation of aspects associated with applying KM to software testing, which addresses the first specific goal of this thesis: Identify the state of the art on KM applied to software testing through a mapping study. Aspects such as purposes of employing KM in software testing, types of knowledge items typically managed in the context of software testing, supporting technologies used, and benefits and problems reported on implementing KM initiatives in software testing, among others, are investigated.

A preliminary version of this mapping study can be referred to in (SOUZA et al., 2013a). This mapping study was updated to include studies published in 2013. Moreover, new studies selected from snowballing of primary study references were included. Snowballing is a process that checks if those studies cite any other relevant

studies, retrieve those studies, and continue that process until cannot find any more relevant studies. Furthermore, direct search to publications of important researchers and research groups were performed.

2.3.1 Study based on a Systematic Mapping

A mapping study provides a broad overview of a research area in order to determine whether there is research evidence on a particular topic. Results of a mapping study may identify suitable areas for performing Systematic Literature Reviews (SLR). Moreover, mapping studies also help identifying gaps in order to suggest areas for future research and provide a map that allows appropriately to position new research activities (KITCHENHAM, 2007; PETERSEN et al., 2008; KITCHENHAM et al., 2011).

In this section a secondary study is presented, i.e. a study that is based on analyzing research papers (referred to as primary studies) (KITCHENHAM et al., 2011). This secondary study comprises both a mapping study and a Systematic Literature Review (SLR). According to Kitchenham et al. (2011), "SLRs are secondary studies [...] used to find, critically evaluate and aggregate all relevant research papers [...] on a specific research question or research topic. The methodology is intended to ensure that the literature review is unbiased, rigorous and auditable. [...] Mapping studies use the same basic methodology as SLR but aim to identify and classify all research related to a broad software engineering topic [...]. They are intended to provide an overview of a topic area and identify whether there are sub-topics with sufficient primary studies to conduct conventional SLR and also to identify sub-topics where more primary studies are needed".

The approach followed in this work starts by performing a mapping study. Then, from the results achieved with the mapping, a SLR is performed, investigating how widespread is the use of ontologies in KM initiatives. This SLR was motivated by the fact that several works in the KM area, such as (O'LEARY, 1998a; BENJAMINS et al., 1998; O'LEARY; STUDER, 2001; STAAB et al., 2001; FENSEL, 2003), advocate in favor of using ontologies for establishing a common conceptualization to be used in KM systems in order to facilitate communication, integration, search, storage and representation of knowledge. Ontologies open the way to move from a document-oriented view of KM to a content-oriented view, where knowledge items are interlinked, combined, and used (STAAB et al., 2001).

2.3.2 Related Work: Secondary study addressing KM in Software Testing

Before accomplishing the secondary study, a tertiary study looking for secondary studies investigating the state of the art in KM in Software Testing was performed. In this, the following search string was used, which was applied in three metadata fields (title, abstract and keywords): ("software testing" OR "software test") AND ("knowledge management") AND ("systematic literature review" OR "systematic review" OR "systematic mapping" OR "mapping study" OR "systematic literature mapping"). The search string was applied in the following electronic databases: *IEEE Xplore, ACM Digital Library, SpringerLink, Scopus, SicenceDirect, Compendex* and *ISI of Knowledge*. Nevertheless, no publication was returned. As any secondary study was found addressing KM in Software Testing, secondary studies that deal with KM and Software Testing separately were investigated.

For the a tertiary study that looks for secondary studies in Software Testing, the following search string was used: ("software testing" OR "software test") AND ("systematic literature review" OR "systematic review" OR "systematic mapping" OR "mapping study" OR "systematic literature mapping"). The same seven electronic databases were searched, returning 149 results. After eliminating duplications and applying the selection criteria, 28 papers presenting secondary studies on software testing were reached. 16 are SLR, while 12 are mapping studies. Different areas related to software testing have been investigated by means of secondary studies. From the 28 secondary studies analyzed, them were grouped in the following categories: (i) Testing of specific software types (13 studies), highlighting Software Product Line Testing (4 studies) and Testing of Software Oriented Architecture and Web Services (3 studies); (ii) Testing Techniques (8 studies), highlighting Search-Based Software Testing (3 studies); (iii) Testing and Software Process (3 studies); (iv) Test Case Prioritization (3 studies); and (v) Others (3 studies), including Testing Automation, Alignment of Software Testing with Requirements, and Test effort reduction. It is worthwhile to point out that 2 studies were classified in more than one category, since they focus on a specific testing technique applied to a specific software type (unit testing approaches for web services, and mutation testing for aspect-oriented programming).

Now, for in the tertiary study that looked for secondary studies in Knowledge Management, the following search string were used: ("knowledge management") AND ("systematic literature review" OR "systematic review" OR "systematic mapping"

OR “Mapping study” OR “systematic literature mapping”). The same seven electronic databases were searched and 239 elements were returned. Duplications were eliminated and search criteria were applied. This resulted in 23 papers presenting secondary studies on KM. 22 are SLR, while only one paper presents a mapping study. From the 23 secondary studies analyzed, them were grouped into the following categories: (i) General Aspects of KM (10 studies), including KM diffusion, KM in organizations, and relationships between KM and other related study areas, such as corporate culture, leadership, innovation, social media, competition and cooperation; (ii) Software Engineering/Software Development (7 studies); (iii) Health (3 studies), (iv) Ontologies and KM (2 studies); and (v) Emergency Management (1 study).

Based on the results from these two investigations by employing the tertiary study, one can say that there is a great diversity of secondary studies in Software Testing and in KM. However, as far as the investigations were concerned, no mapping study or SLR combining these two areas was found. It is also interesting to note that the 51 secondary studies that were analyzed have been published since 2008.

2.3.2.1 Research Method for the Mapping

The research method for this mapping was defined based on the guidelines for systematic literature reviews given in Kitchenham (2007). A protocol for the entire process of the mapping study was developed. The purpose of this review protocol is to support researchers in avoiding bias in conducting the review. The systematic mapping protocol involves three main essential process phases (KITCHENHAM, 2007): (i) **Planning**: refers to the pre-review activities, and aims at establishing a review protocol defining the research questions, inclusion and exclusion criteria, sources of studies, search string, and mapping procedures; (ii) **Conducting**: regards searching and selecting the studies, in order to extract and synthesize data from them; (iii) **Reporting**: is the final phase and aims at writing up the results and circulating them to potentially interested parties. In this phase the findings of the systematic mapping study are used to answer the research questions. The main steps followed when performing in this systematic mapping study are described in detail following.

Research Questions: This mapping aims at answering the following research questions presented in Table 2.2:

Table 2.2 - Research questions and their rationales

N^o	**Research question**	**Rationale**
RQ1	When and where have the studies been published?	The topic of this mapping study seems to be broad and new. This research question aims at giving an understanding on whether there are specific publication sources for these studies, and when they have been published.
RQ2	From the software testing perspective, what aspects have been focused in the research?	Investigates which aspects of software testing have been the subject of KM initiatives. This information can help to identify which aspects of software testing have gained more attention when applying KM in software testing.
RQ3	From the KM perspective, which topics have been focused?	Similarly to the previous, this research question investigates the KM sub-topics that have been more explored when applying KM to software testing.
RQ4	What types of research have been done?	As pointed out by Wieringa et al. (2006) and Petersen et al. (2008), different types of research have been presented in scientific papers (e.g., solution proposal, validation research, experience papers, opinion paper and evaluation research). This research question investigates which type of the research is reported in each selected study. This is an important question, since it can be used to evaluate the current maturity stage of the area.

Continues

Table 2.2 - Conclusion

N^o	**Research question**	**Rationale**
RQ5	What are the problems reported by software organizations related to knowledge about software testing?	Provides an overview of the main problems reported by organizations related to the lack of knowledge about software testing. The goal of this question is to point out the problems that have motivated the research in this field.
RQ6	What are the purposes of employing KM in software testing?	This question complements the previous one, looking for the purposes declared in the studies for managing software testing knowledge. This is important to point out why such studies have been accomplished.
RQ7	What are the types of knowledge items typically managed in the context of software testing?	Investigates the types of knowledge items that have been managed in software testing. This information is important, since it gives a roadmap to define which types of knowledge have been considered more important in software testing.
RQ8	What are the technologies used to provide KM in software testing?	Highlights the main technologies currently used to provide KM in software testing. This is useful for researchers and practitioners that intend to accomplish new initiatives of KM in software testing, as well as to guide future research towards new technologies in order to fill the existing gaps.

Continues

Table 2.2 - Conclusion

N^o	**Research question**	**Rationale**
RQ9	What are the main conclusions reported regarding applying KM in software testing?	Compiles the main conclusions reported on the studies regarding KM in software testing. This information is useful for evaluating the actual benefits of the current studies of KM in software testing, as well as to point out problems that remain and, thus, need to be subject of further research.

Inclusion and Exclusion Criteria: The selection criteria are organized in one inclusion criterion (IC) and five exclusion criteria (EC). The inclusion criterion is: (IC1) The study discusses KM applied to software testing. The exclusion criteria are: (EC1) The study does not have an abstract; (EC2) The study is just published as an abstract; (EC3) The study is not written in English; (EC4) The study is an older version (less updated) of another study already considered; and (EC5) The study is not a primary study, such as editorials, summaries of keynotes, workshops, and tutorials.

Sources: The search was applied in seven electronic databases that were considered the most relevant according to (DYBA et al., 2007). They are:

IEEE Xplore (*http://ieeexplore.ieee.org*)
ACM Digital Library (*http://dl.acm.org*)
SpringerLink (*http://www.springerlink.com*)
Scopus (*http://www.scopus.com*)
Science Direct (*http://www.sciencedirect.com*)
Compendex (*http://www.engineeringvillage2.org*)
ISI of Knowledge (*http://www.isiknowledge.com*)

Keywords and Search String: The search string considered two areas, Software Testing and KM (Table 2.3), and it was applied in three metadata fields (title, abstract and keywords). The search went through syntactic adaptations according to particularities of each source.

Table 2.3 - Keywords of the Search String of the Mapping

Areas	Keywords
Software Testing	"Software Testing", "Software Test"
KM	"Knowledge Management", "Knowledge Reuse", "Packaging Experience"
Search String: ("Software Testing" OR "Software Test") AND ("Knowledge Management" OR "Knowledge Reuse")	

Data storage: The publications returned in the searching phase were cataloged and stored appropriately. A data extraction form was developed to gather all relevant data from the identified studies (e.g., id and bibliographic reference). This catalog helped in the studies classification and analysis procedures.

Assessments: Before conducting the mapping, the protocol was tested. This test was conducted in order to verify its feasibility and adequacy, based on a pre-selected set of studies considered relevant to investigation. In particular, in order to elaborated the search string, the set of search terms were devised in an iterative fashion, i.e. an initial set of terms were elaborate and iteratively improved this set until all relevant pre-selected studies were found. The review process was conducted just this author of this thesis and the other two carried out its validation. Approximately 36% of the studies were analyzed using two different samples.

2.3.2.2 Data extraction and synthesis

The search process considered studies published until November 2013. As a result of searching the selected sources, a total of 345 publications were returned, out of which 55 from IEEE Xplore, 67 from Compendex, 72 from Scopus, 2 from Science Direct, 4 from ACM Digital Library, 139 from SpringerLink, and 6 from ISI of Knowledge. Then a selection process comprising five stages were followed, as shown in Figure 2.3.

In the 1^{st} stage, duplications were eliminated (publications that appear in more than one source), achieving 261 publications (reduction of approximately 24%). In the 2^{nd} stage, the selection criteria (inclusion and exclusion criteria) were applied over title, abstract and keywords, resulting in 37 papers (reduction of approximately 86%). 7 papers were eliminated by EC1 (The study does not have an abstract); 2 by EC3 (The study is not written in English); 15 by EC5 (The study is not a primary study); and 200 for not satisfying IC1 (The study discusses KM applied to software testing).

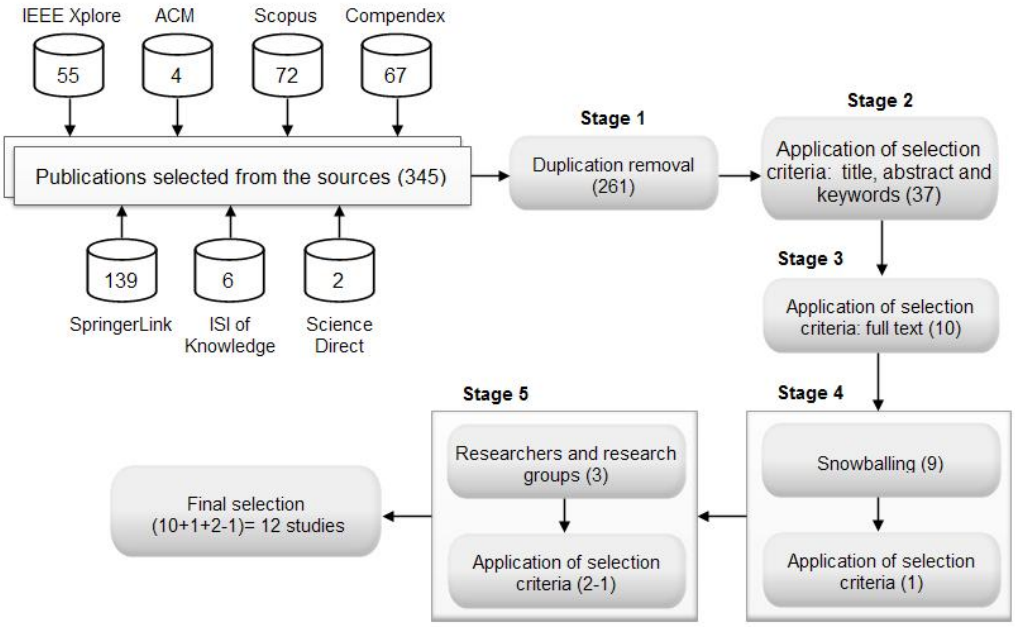

Figure 2.3 - Search and selection mapping process

In the 3^{rd} stage, the selection criteria were applied considering the full text, resulting in a set of 10 studies (reduction of approximately 73%). 2 papers were eliminated by EC4 (The study is an older version of another study already considered); and 25 papers were eliminated for not satisfying IC1 (The study discusses KM applied to software testing).

Over these 10 considered relevant, in the 4^{th} stage the snowballing was performed, which resulted in 9 papers. After applying the selection criteria, only 1 paper remained. Criterion EC3 eliminated 4 papers (The study is not written in English); 1 by EC5 (The study is not a primary study); and 3 papers for not satisfying IC1 (The study discusses KM applied to software testing). These last 3 papers were eliminated after reading their full versions.

Finally, from the 11 papers selected until then, in the 5^{th} stage, publications authored by the researchers and research groups involved in these studies were analysed. To conduct that, their personal pages were searched, their entries in The DBLP Computer Science Bibliography, as well as other publications authored by them in the digital libraries that we used as sources for this mapping. 3 papers were selected from the same research group. From these 3 papers, 1 of them was eliminated by EC4 (The study is an older version of other study already considered). Moreover, one of the papers of the resulting set of the 3^{rd} stage was also eliminated by EC4, since

one of the papers selected from the research group were newer and more complete.

As a final result, 12 studies were analyzed (9 from the sources, 1 from snowballing, and 2 from direct search to researchers and research groups). Table 2.4 summarizes the stages and their results. It shows the progressive reduction of the number of studies throughout the selection process. Table 2.5 presents the bibliographic reference of the selected studies plus an identifier (#id) for each paper. Throughout the remainder of this paper, these identifiers to refer to the corresponding publication were used.

Table 2.4 - Results from the selection stages

Stage	**Applied Criteria**	**Analyzed Content**	**Initial Number of Studies**	**Final N. of Studies**	**Reduction (%)**
1^{st}	Duplicate Removal	Title, abstract and keywords	345	261	24.3%
2^{nd}	IC1, EC1, EC3 and EC5	Title, abstract and keywords	261	37	85.8%
3^{rd}	IC1 and EC4	Full Text	37	10	73.0%
4^{th}(a)	Snowballing, EC3 and EC5	Title, abstract and keywords	9(added by snowballing)	4(added by snowballing)	55.6%
4^{th}(b)	Snowballing, IC1	Full Text	4(added by snowballing)	1(added by snowballing)	75.0%
5^{th}	Research Group, EC4	Full Text	3(added by research groups)	2(added by research groups) - 1(selected from the sources)	33.3%
Final Result			345 + 9 (snowballing) + 3 (research groups) = 357	9 + 1 (snowballing) + 2 (research groups) = 12	96.6%

Table 2.5 - Selected Studies

ID	**Bibliographic Reference**
#1	Liu, Y.; Wu, J.; Liu, X.; Gu, G. Investigation of Knowledge Management Methods in Software Testing Process. *In: International Conference on Information Technology and Computer Science*, Kiev, Ukraine, vol. 2, 90-94, 2009.

Continues

Table 2.5 - Conclusion

ID	Bibliographic Reference
#2	Wei, O. K.; Ying, T. M. Knowledge Management Approach in Mobile Software System Testing. *In: International Conference on Industrial Engineering and Engineering Management*, Singapore, 2120-2123, 2007.
#3	Xu-Xiang, L.; Wen-Ning, Z. The PDCA-based software testing improvement framework. *In: International Conference on Apperceiving Computing and Intelligence Analysis (ICACIA)*, Chengdu, China, 490-494, 2010.
#4	Abdullah, R.; Eri, Z. D.; Talib, A. M. A Model of Knowledge Management System in Managing Knowledge of Software Testing Environment. *5th Malaysian Conference in Software Engineering (MySEC)*, Johor Bahru, Malaysia, 229-233, 2011.
#5	Li, X.; Zhang, W. Ontology-based Testing Platform for Reusing. *In: International Conference on Internet Computing for Science and Engineering*, Henan, China, 86-89, 2012.
#6	Desai, A.; Shah, S. Knowledge Management and Software Testing. *In: International Conference and Workshop on Emerging Trends in Technology (ICWET)*, Mumbai, India, 767-770, 2011.
#7	Andrade, J.; Ares, J.; Martínez, M.; Pazos, J.; Rodríguez, S.; Romera, J.; Suárez, S. An architectural model for software testing lesson learned systems. *Information and Software Technology*, vol. 55, Issue 1, 18-34, 2013.
#8	Nogeste, K.; Walker, D.H.T. Using knowledge management to revise software-testing processes. *Journal of Workplace Learning*, vol. 18, Issue 1, 6-27, 2006.
#9	Janjic, W.; Atkinson, C. Utilizing software reuse experience for automated test recommendation. *In: Proceedings of the 8th International Workshop on Automation of Software Test (AST)*, San Francisco, USA, 100-106, 2013.
#10	Kerkhof, C.; Ende, J.; Bogenrieder, I. Knowledge Management in the Professional Organization: A Model with Application to CMG Software Testing. *Knowledge and Process Management*, vol. 10, Issue 2, 77-84, 2003.

Continues

Table 2.5 - Conclusion

ID	Bibliographic Reference
#11	Vegas, S.; Basili, V.R. A Characterization Schema for Software Testing Techniques. *Empirical Software Engineering*, vol. 10, Issue 4, 437-466, 2005.
#12	Vegas, S.; Juristo, N.; V.R. Basili. Packaging experiences for improving testing technique selection. *The Journal of Systems and Software*, vol. 79, Issue 11, 1606-1618, 2006.

2.3.2.3 Classification scheme

For conducting a systematic mapping, a classification scheme needs to be defined (PETERSEN et al., 2008). Different facets were considered, one for each research question. The exception is the last question (RQ9), for which were collected unstructured data without a predefined classification. Only the main findings found were considered, as benefits and problems related with the implementation of KM in software testing reported by the selected studies. The categories comprising the other facets were defined following two approaches: (i) based on categories already considered in the literature; and (ii) taking the selected studies into account. Following, the categories of these facets are presented.

Research focus from the software testing perspective (RQ2): Studies on KM in software testing have focused on different aspects of software testing. Based on the selected studies, six main categories were considered, described below. Note that, according to classification scheme, one study can span more the one research focus regarding the testing perspective.

- **Testing Process:** the focus is on managing knowledge in the context of a given testing process.
- **Test Case:** the focus is on managing knowledge about test cases, for supporting, e.g., test case reuse.
- **Testing Phase:** the study discusses the application of KM in a specific level test, such as unit testing, system testing, integration testing, regression testing.
- **Testing Technique:** the focus is on managing knowledge about testing

techniques, aiming at helping testers to select better suited testing technique for designing test cases.

- **Third Party Testing:** the study discusses KM applied to situations in which testing is accomplished by a third party (e.g. outsourcing).

- **General:** this category is used to classify those papers that discuss KM in software testing in general, without focusing in any specific aspect of software testing.

Research focus from the KM perspective (RQ3): Similarly to the previous, studies on KM in software testing also focus on different aspects of KM. Based on the selected studies, the categories described below were considered. Again, a study can span more the one research focus regarding the KM perspective.

- **Knowledge Management Model:** the study discusses a model for managing knowledge, considering knowledge processes, and, eventually some aspects of it, such as knowledge carriers.

- **Knowledge Representation:** the study discusses aspects related to how to represent testing knowledge.

- **Knowledge Packing:** the study goes beyond aspects related to knowledge representation, focusing on how to pack it. Studies classified in this category are also classified in the previous.

- **Knowledge Capturing:** the study addresses aspects related to how to acquire and store testing.

- **Knowledge Elicitation:** the study goes beyond aspects related to knowledge capturing, discussing also ways to elicit knowledge from experts. Studies classified in this category are also classified in the Knowledge Capturing category.

- **Knowledge Retrieval:** the study addresses aspects concerning retrieval of testing knowledge. In this case, the user is responsible for searching knowledge items.

- **Knowledge Dissemination:** regards pro-actively disseminating testing knowledge.

- **Knowledge Evolution:** the study approaches aspects related to the evolution of the testing knowledge already stored, such as evaluation and maintenance.

- **Knowledge Management Systems (KMS):** the study discusses aspects related to providing automated support for managing testing knowledge by means of a system. Studies classified in this category may be describing an actual system, as well as an architectural model or general features of a KMS.

Research type (RQ4): This facet captures the research approach used in the studies. It is general and independent from a specific focus area, and thus an existing classification was adopted: the one proposed by Wieringa et al. (2006), and revisited by Petersen et al. (2008) to become more general. Based on the selected studies, some of its categories were disregarded, since none of the selected studies were classified in those categories. The categories used in this mapping are summarized below. As pointed out by Wieringa et al. (2006), studies can span more than one category, although some combinations are unlikely.

- **Solution Proposal:** In this research approach, the study proposes a solution for a problem and argues for its relevance, without a full-blown validation. The solution must be novel, or at least a significant improvement of an existing one. A proof-of-concept may be offered by means of a small example, a sound argument, or by some other means.

- **Validation Research:** In this research approach, the study investigates the properties of a proposed solution that has not yet been implemented in practice. It may have already been proposed elsewhere by the author or by someone else. The investigation uses a thorough, methodologically sound research setup. Possible research methods include, among others, experiments, prototyping and simulation.

- **Evaluation Research:** In this type of research, the study discusses the implementation of a technique in practice, and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). The novelty of the technique is not necessarily a contribution of the study. However, if no industry cooperation or real world project is mentioned, then the study cannot be considered an evaluation research (PETERSEN et al., 2008).

Reported Problems (RQ5): The categories for this facet are based on the main problems related to knowledge about software testing reported in the selected studies. Five main categories of problems were identified, namely: (i) Barriers in transferring testing knowledge; (ii) Loss of testing knowledge; (iii) Low reuse rate of testing knowledge; (iv) Testing knowledge is not properly shared; and (v) Testing knowledge is not properly considered for planning the testing process (including human resource allocation to testing activities). Studies can span more than one category.

Purposes (RQ6): In this facet, interest is to learn the organizations´ purposes, when employing KM in software testing. Five main categories of such purposes were identified: (i) Reuse of knowledge related to software testing, (ii) Support for decision making, (iii) Cost reduction, (iv) Competitive advantages, and (v) Organizational learning. Studies can span more than one category.

Types of knowledge (RQ7): This facet concerns the types of knowledge that have been dealt with in the research. In this case, the classification proposed by Nonaka and Takeuchi (1997) were adopted, that distinguishes between tacit and explicit knowledge. Moreover, for explicit knowledge, based on the selected studies, the types of knowledge that have been managed were identified into the following categories: Software Artifacts (essentially test cases), Classification Schemes (for testing techniques), Lessons Learned (about software testing), and Knowledge Packages. Since some studies did not indicate a specific type of knowledge, in such cases it is considered that they fall in the General category. Studies can span more than one category.

Technologies used (RQ8): This facet discusses the technologies that have been used or proposed to support managing software testing knowledge. Based on the selected studies, the following categories were considered: (i) Ontologies, (ii) Yellow Pages (or Knowledge Maps), (iii) Agents, (iv) Recommendation Systems, (v) Data warehouse, and (vi) Conventional Technologies. This last category mentions IT conventional technologies, such as databases, intranets, and Internet. Studies can span more than one category, but can also fit any, if the study does not explicitly approach this issue.

2.3.3 Limitations of this mapping

The mapping has some limitations. Due to the fact that the study selection and data extraction steps were performed by just this author of this thesis, some subjec-

tivity may have been embedded. To reduce this subjectivity, the other two persons (advisers) performed these same steps on a random sample (including about 36% of the studies). The results of each reviewer were then compared in order to detect possible bias. Moreover, terminological problems in the search strings may have led to missing some primary studies. In order to minimize these problems, previous simulations in the selected databases were performed. Any specific conference proceedings, journals, or the grey literature (technical reports and works in progress) were searched. Thus, only studies indexed by the selected electronic databases were worked. The exclusion of these other sources makes the review more repeatable, but possibly some valuable studies may have been left out of our analysis.

2.3.4 Results

The mapping study was performed according to the steps described in Section 2.3.2.1. This section presents the results for each of the research questions defined. To answer the questions, a form with the #id of the paper was used, with its bibliographic reference, and including, the facets of the classification schema aforementioned. This form was used to extract the answers for each research question.

1) Classification by publication year and source (RQ1)

In order to offer a general view of the efforts in the area of KM in Software Testing, a distribution of the 12 selected papers over the years is shown in Figure 2.4. As this figure suggests, KM in software testing is recent, occurring basically in 2003.

The selected studies were published in three main vehicles: Journals, Conferences and Workshops. Conferences have been the main forum for presenting KM in software testing, encompassing 50% (6 studies in 12). Journals are the publication forum of 41.7% (5 out of 12). Finally, only one was presented in a Workshop (8.3%). Table 2.6 presents the publication sources of all the selected studies, their types, and their (#id). It is worth pointing out that 12 different publication sources were identified, one for each study, showing that currently there is no well-established forum for discussing the topic. Publications vehicles in areas such as IT, Software Engineering and Knowledge Management seem to be more receptive for presenting studies in the area.

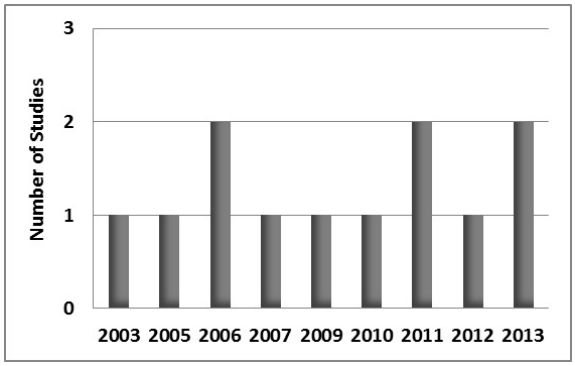

Figure 2.4 - Distribution of the selected studies over the years

Table 2.6 - Publication Sources

Publication Sources	**Type**	**#ID**
International Workshop on Automation of Software Test (AST)	Workshop	#9
International Conference on Information Technology and Computer Science (ITCS)	Conference	#1
Industrial Engineering and Engineering Management	Conference	#2
Apperceiving Computing and Intelligence Analysis (ICACIA)	Conference	#3
Malaysian Conference in Software Engineering (MySEC)	Conference	#4
International Conference on Internet Computing for Science and Engineering (ICICSE)	Conference	#5
International Conference and Workshop on Emerging Trends in Technology (ICWET)	Conference	#6
Information and Software Technology	Journal	#7
Journal of Workplace Learning	Journal	#8
Knowledge and Process Management	Journal	#10
Empirical Software Engineering: An International Journal	Journal	#11
Journal of Systems and Software	Journal	#12

2) Research focus from the software testing perspective (RQ2)

Table 2.7 shows the distribution of the studies according to the research focus from

the software testing perspective. Most of the papers address a specific aspect of software testing (8 out of 12 - 66.7%) for providing KM facilities. Test case reuse (3), selection of testing techniques (2), and third party testing (2) are aspects that have also received attention. In particular, test case reuse has been the main focus of the recent research, as well as some focus on a specific testing phase. For instance, in #8, Nogeste and Walker (2003) proposes a KM-based regression testing process, while #9, Janjic and Atkinson (2013) discuss KM-based support for reusing class and unit test cases. On the other hand, 4 studies (33.3%) discuss KM applied to software testing in general, i.e. without focusing on any specific aspect of software testing.

Table 2.7 - Research focus from the software testing perspective along the years

Research Focus	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Testing Process			#8			#3				2
Test Case							#6	#5	#9	3
Test Phase			#8	#2					#9	3
Testing Technique		#11	#12							2
Third Party Testing				#2		#3				2
General	#10				#1		#4		#7	4

3) Research focus from the KM perspective (RQ3)

With respect to KM, as Table 2.8 shows, the great majority of the studies discussed aspects related to the KM process as a whole (KM Model) or focusing on one of its activities: knowledge representation (6), capture (6), and retrieval (5). Moreover, most of the papers (75%) describe KM systems to provide KM-based support for software testing. The study #7 is the one that covers the largest number of KM activities.

Table 2.8 - Distribution over research focus regarding the KM perspective

Research Focus	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Knowledge Management Model	#10		#8				#4,#6	#5	#7	6
Knowledge Representation		#11	#12	#2	#1			#5	#7	6
Knowledge Packing			#12							1

Continues

Table 2.8 - Conclusion

Research Focus	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Knowledge Capturing		#11	#8,#12	#2	#1				#7	6
Knowledge Elicitation				#2					#7	2
Knowledge Retrieval		#11		#2	#1				#7, #9	5
Knowledge Dissemination									#7, #9	2
Knowledge Evolution								#5		1
Knowledge Management Systems (KMS)			#8	#2	#1	#3	#4, #6	#5	#7, #9	9

4) Research Type (RQ4)

Table 2.9 presents the distribution according to the research types. Considering the categories defined by Wieringa et al. (2006), research in KM applied to software testing is dominated by Solution Proposals. All the 12 studies propose some solution for KM in software testing. In addition to presenting a solution proposal, most of them also discuss some sort of evaluation: 7 studies also report an Evaluation Research (58.3%), while 3 studies report a Validation Research (see Figure 2.5). The 7 studies with an Evaluation Research discuss a practical implementation, and its consequences. There is a great interest in implementing the proposals in practice and especially in studies of real cases.

Table 2.9 - Distribution over research type

Research Type	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Validation Research		#11			#1				#9	3
Evaluation Research	#10		#8,#12			#3	#6	#5	#7	7
Solution Proposal	#10	#11	#8,#12	#2	#1	#3	#4,#6	#5	#7,#9	12

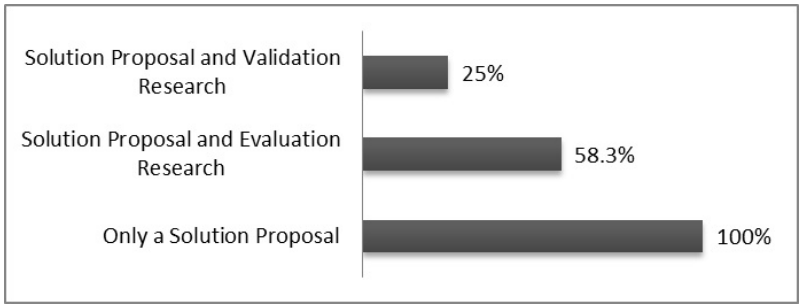

Figure 2.5 - Distribution over research type

5) Reported problems (RQ5)

Table 2.10 shows the distribution over the years considering the problems reported by software organizations related to knowledge about software testing. These problems were regarded as a motivation to perform research in KM to software testing. Among problems categories, "Low Reuse Rate Knowledge" has the largest representativeness (9 out of 12, corresponding to 75%). Software testing, in general, can involve reusing modules, test cases, components, and experiences. However, testing teams, generally, do not reuse or take advantage on the knowledge acquired or the experience gained. Therefore, the same mistakes are repeated, even though there are individuals in the organization with the knowledge and experience required to prevent this (ANDRADE et al., 2013). Another category with a high percentage is "Barriers in Knowledge Transfer" with 8 out of 12, corresponding to 66.6%. It stands out because transfer of organizational knowledge can be quite difficult to achieve. This occurs also because most of the knowledge in organizations is tacit, that is, derived from experience, and it becomes difficult to articulate.

Table 2.10 - Distribution of related problems (motivation)

Problems	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Loss of testing knowledge	#10			#2	#1		#6	#5		5
Barriers in testing knowledge transfer			#8	#2	#1	#3	#4,#6	#5	#7	8
Low reuse rate of testing knowledge	#10	#11	#12		#1		#4,#6	#5	#7,#9	9
Problems related to testing knowledge sharing	#10				#1		#4,#6		#7	5

Continues

Table 2.10 - Conclusion

Problems	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Testing knowledge not considered in test planning			#8		#1		#6	#5		4

Figure 2.6 shows the number of studies per category considering the problems reported as a motivation to conduct the research.

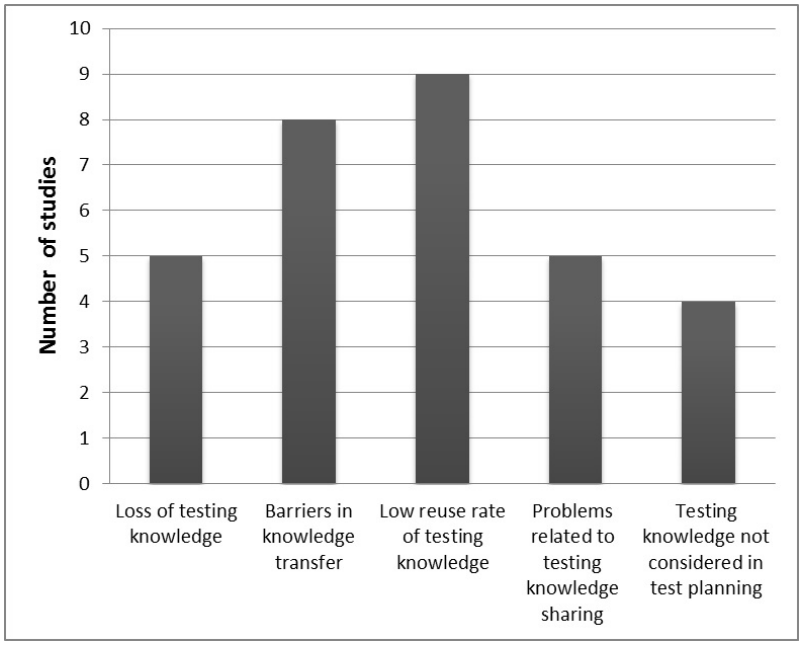

Figure 2.6 - Percentage of the selected studies per problems reported

6) Purposes to employ KM in software testing (RQ6)

Table 2.11 presents the distribution over the years considering the organizations´ purposes in managing software testing knowledge. The categories "Knowledge Reuse" (8 studies - 66.6%), "Organizational Learning" (6 studies - 50%) and "Competitive Advantages" (6 studies - 50%) have the largest representativeness. It is possible to highlight that some purposes that were identified are strongly related. For instance, lessons learned are means to promote both knowledge reuse and organizational learning. Thus, studies that reported that one of the purposes of applying KM in software testing is registering and disseminating lessons learned (5 studies - 41.6%) were considered in both categories. Knowledge reuse, in turn, helps increasing

test effectiveness and thus leads to competitive advantages and cost reduction.

Table 2.11 - Distribution of purposes

Purposes	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Knowledge Reuse			#8	#2	#1	#3	#4, #6	#5	#7	8
Competitive Advantages	#10		#8		#1	#3		#5	#7	6
Organizational Learning			#8, #12			#3	#4		#7,#9	6
Decision Making		#11		#2			#4, #6		#7	5
Cost Reduction				#2		#3			#7	3

Figure 2.7 shows the number of studies per category, considering the organizations´ purposes in managing software testing knowledge.

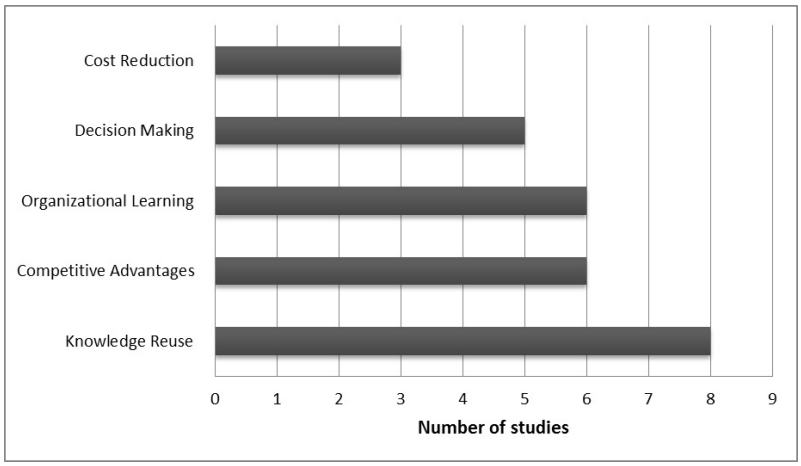

Figure 2.7 - Percentage of the selected studies per purposes reported

7) Types of knowledge being managed (RQ7)

The classification proposed by (NONAKA; TAKEUCHI, 1997) distinguishing between tacit and explicit knowledge was adopted. In the 12 selected studies, both are considered. Tacit knowledge is discussed in 8 studies (66,7%). They mention that it can be acquired from discussions, experiences from project members, questionnaires, expert network, personalized training and communications. On the other hand, all the 12 studies have discussed explicit knowledge. Figure 2.8 shows the percentage of studies per category.

Moreover, for explicit knowledge, based on the 12 selected studies, the types of knowledge that have been managed were identified. Table 2.12 presents the distribution over the years. 50% of studies do not specify a specific type of knowledge, in such cases, considered that they fall into the General category. Out of all the identified types, Software Artifacts are the most commonly cited category (4 out 12 studies, corresponding to 33.3%) (Figure 2.9). Moreover, Test Cases are the software artifacts managed in the 4 studies classified in this category. According to Li and Zhang (2012), the quality of test case has a direct and significant impact on the software testing quality, and hence in the software quality. Thus, introducing the reuse theory into test cases seems to be effective and can promote test case design efficiency.

Table 2.12 - Distribution of explicit knowledge

Explicit knowledge	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Knowledge Packages			#12							1
Lessons Learned									#1	6
Classification Schema		#11	#12							2
Software Artifacts			#8				#6	#5	#9	4
General	#10			#1	#2	#3	#4, #6			6

Figure 2.8 - Percentage of studies per category

8) Technologies used (RQ8)

Technologies used to implement KM in software testing are shown in Table 2.13. 6 different technologies in this context were found. It is worthwhile to point out that 3 studies do not address this issue (#3, #11, #12), representing 25%. In #3, Xu-Xiang and Wen-Ning (2010) proposes a process, but does not discuss technologies; in #11 and #12, a classification scheme is proposed, and this scheme is used without automated support, and without discussing aspects related to technology. The great

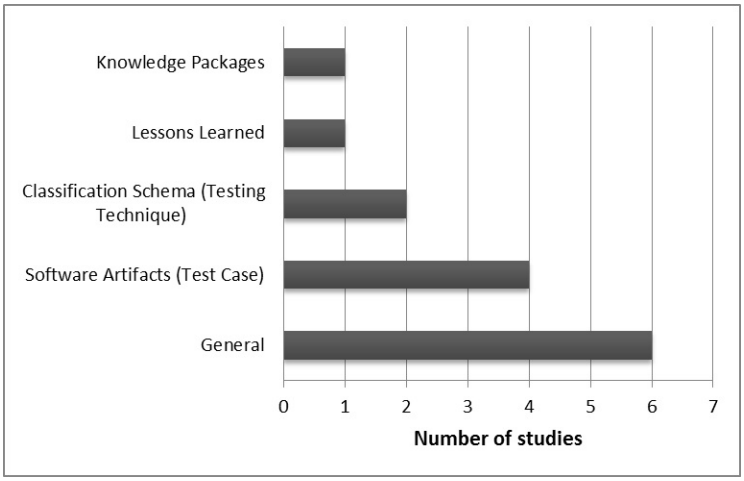

Figure 2.9 - Types of knowledge managed

majority of the studies use conventional technologies (7 in 12 studies, corresponding to 58.3%). This category concerns IT conventional technologies, such as databases, intranets, and Internet. Yellow Pages (or Knowledge Maps) are discussed in 25% of the studies (3 in 12). Figure 2.10 shows the number of studies per category, considering the technologies used to implement KM in software testing.

Table 2.13 - Distribution of technologies used

Problems	2003	2005	2006	2007	2009	2010	2011	2012	2013	Total
Agents									#9	1
Data Warehouse							#6	#5		2
Recommendation Systems								#7,#9		2
Ontologies					#1			#5		2
Yellow Pages (or Knowledge Maps)	#10			#1				#5	#7	3
Conventional Technologies	#10		#8	#2			#4,#6	#5	#7	7
Study does not address this issue		#11	#12			#3				3

Organizations can extract testing knowledge and create the testing knowledge base and then establish the reusable test knowledge repository. This repository may be a data warehouse as cited in #5 and #6 (16.7% of the studies). Moreover, the representation of the knowledge structure can be in accordance with an ontology representation (#1, #5) (16.7%).

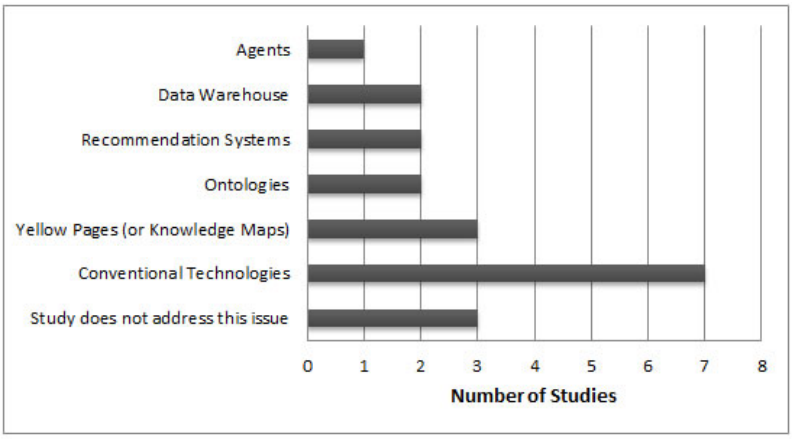

Figure 2.10 - Technologies used to implement KM

9) Benefits and Problems reported on the implementation of KM initiatives in software testing (RQ9)

As we can see from this mapping, there are many benefits of implementing KM in organizations for managing software testing knowledge:

- **Selection and application of better suited techniques, methods and test cases.** Experience plays a key role in testing, and managing past experience helps to effectively tailor the techniques and methods to the ongoing project. Some of these techniques, such as White-box Testing, Black-box Testing or Defect-based Testing, depend on the knowledge, experience and intuition of the tester.

- **Cost reduction.** In the testing context, cost is strongly related with time. Tester´s experience is crucial for designing test cases and regression test selection. A good selection of test cases minimizes not only costs but also time.

- **Test effectiveness increase.** Knowledge and experience about the domain and the system under test are essential for increasing test effectiveness. This helps testers to decide on which techniques to use, and to select test cases, among others.

- **Competitive advantages.** In organizations, KM is now seen as a strategic factor and knowledge is also recognized as one of the main sources of

cost savings and competitive advantage. The ability to transfer best practices in the organization is a means to build competitive advantage through the appropriation from scarce knowledge.

Although KM in software testing brings many benefits, there are also problems:

- **Employees are normally reluctant to share their knowledge:** Experiences are grasped by few people only and do not turn into public knowledge. This fact disables knowledge transfer in testing.
- **Increased workload:** Shortage of time is a potential risk to incorporate the principles of KM in software testing, because knowledge sharing can imply in increasing the employee workload and costs.
- **KM systems are not appropriate yet:** There are many difficulties in implementing knowledge acquisition, coding, storage and searching functionalities effectively in KM systems, because it involves all the problems mentioned above, as time and interest of the employees.

2.3.4.1 Discussion of reviewed studies

In this section, some important findings in the mapping study are discussed. From the results achieved with the mapping, reports on ontology-based approaches for KM in software testing are analyzed, by means of a Systematic Literature Review (SLR).

Software has become more and more widespread and indispensable in critical and complex application domains, making testing processes increasingly important and complex, and calling for greater quality. Furthermore, testing process is a knowledge-intensive activity. In this context, knowledge and experience can be very useful, and testing team members can make use of this experience. However, studies have reported that the greatest problem is knowledge reuse within organizations. The main issue is that knowledge is retained with a single individual and therefore becomes more difficult to raise this knowledge to the organizational level. Even when some KM strategy is applied, it is not always feasible to achieve organizational learning because the employees are reluctant to share their knowledge as they feel that retaining this knowledge is an advantage over their colleagues (ANDRADE et al., 2013). All these problems are pointed out as purposes to applying KM in software testing, in particular knowledge reuse.

Out of the 12 studies analyzed in this mapping, KM in software testing has focused on different aspects. Reuse of test cases is a perspective that has received more attention. Software testing in general involves reusing of modules and test cases, because most of the software can be reused from time to time. Reusing knowledge already present in existing test cases has a potential to significantly reduce software development costs and time consumption (DESAI, 2011; JANJIC; ATKINSON, 2013).

With respect to the KM perspective, the great majority of the studies discuss aspects related to implementing a KM process as a whole or focusing on one of its activities. Most of the studies discuss aspects related to providing automated support for managing testing knowledge by means of a system. These studies present a KMS as an actual system, as a prototype or as well as an architectural model. On the other hand, considering the research type performed by the 12 papers presented, Solution Proposal is most prominent; all 12 studies present a proposal. Moreover, 7 studies discuss the implementation of an approach in practice. There is a significant interest in implementing the proposals in practice and especially in real world scenarios.

With respect to the types of knowledge, both tacit and explicit have been considered important. Most of the studies identify that tacit knowledge is more difficult to acquire, as part of personal experiences by the members of the test team. According to Nonaka and Takeuchi (1997), as expected, tacit knowledge really is hard to be acquired, and it requires good strategies to acquire and process this knowledge; however, it is more valuable. In papers #4, #7, #10, for instance, although dealing with the two types of knowledge (tacit and explicit), the authors also emphasize the importance to capture the tacit knowledge and treat it.

During the mapping it was possible to infer that much of the explicit knowledge was related to reuse of test cases. According to Liu et al. (2009), more detailed information on test cases can provide a greater learning. As test cases evolve in applications, they may be changed for a variety of reasons. Thus an efficient and effective KM process can help in evaluating the impact and in conducting changes of the test cases.

With respect to the technologies used to implement KM, Knowledge Maps or Yellow Pages seem to have good results where test engineers want to find the right experts for helping them. A Knowledge Map contains information about experiences that an employee possesses. It is a stock catalogue about knowledge (LIU et al., 2009). In #1, for instance, a KM model is created and one of the components of this model is a Knowledge Map repository, considered the kernel of the system. The system

identifies, by means of statistics, the staff with knowledge, and that will improve the culture of knowledge-sharing in the enterprise. In #5, Li and Zhang (2012) present a Knowledge Management Model and one of the elements of this model is also a knowledge map.

On the other hand, Data Warehouse, Recommendation System and Ontologies are also discussed. In special, ontologies minimize ambiguity and imprecision in interpreting shared information (KIM, 2000). Ontologies are considered a key enabling technology for KM (KIM, 2000). They provide a shared and common understanding of a domain that can be communicated between people and application systems. Their use offers an opportunity for improving KM capabilities in large organizations (DAVIES et al., 2003). In ontology-based KM systems, ontologies are mainly used (ABECKER; ELST, 2004): (i) to support knowledge search, retrieval, and personalization; (ii) to serve as basis for knowledge gathering, integration, and organization; and (iii) to support knowledge visualization. More specifically, domain ontologies are used to structure the content area of documents and providing background knowledge for inferences.

Three studies (#1, #5, and #7) discuss ontologies in some aspect. However, only two (#1 and #5) indeed, use ontology. This seems to be a problem, since, as pointed out by Staab et al. (2001), ontologies are capable of binding KM activities together, allowing a content-oriented view of KM. One possible explanation for this low number of studies addressing ontology-based KM initiatives in software testing is the fact that developing ontology is a hard task, especially in complex domains, as is the case of software testing.

2.3.4.2 SLR in ontology-based KM initiatives

From these 12 studies, two studies used ontology, and only one used ontology on software testing domain. Thus, looking into Ontology-based KM in software testing, a SLR was performed. The following research questions about the two ontologies found (Table 2.14) and after is presented an outline of the two studies (#1 and #5).

Table 2.14 - Research questions and their rationales

N^o	**Research question**	**Rationale**
RQ1	What kinds of ontologies have been used?	There are different classifications of ontologies. In this research question is necessary to know what kind of ontology have been used.
RQ2	What are the purposes of using ontologies in those initiatives?	This question investigates the main purposes of using ontologies. This information is useful for evaluating the actual benefits of the use of ontologies in KM applied to software testing.
RQ3	What are the languages or formalisms used to create those ontologies?	To represent a shared conceptualization, a language of representation is required. This question investigates which languages or formalisms have been used to create those ontologies.
RQ4	When are the ontologies used (at development time and/or at run time)?	This question investigates when ontologies are employed (at development time and/or at run time).

Liu et al. (2009) (#1), from experiences acquired from many software testing projects in the Beijing University of Aeronautics and Astronautics (BUAA), concluded that it is very important to carry out KM in software testing, and they propose a KM model for software testing. In this model, ontology is used for structuring a knowledge base and for searching and sorting. From this model, a prototype of a KM system was designed and integrated into a test platform called QESuite2.0, which has been developed in BUAA. In #1, although Liu et al. (2009) say that ontology "classifies concepts of software testing, describes relation restriction between concepts, and constructs detailed knowledge database", there is no mention to the use of an ontology of software testing. In fact the model presented as the "Knowledge Ontology" contains the following concepts: *Knowledge Carriers, Document, Reference, Staff, Content, Project* and *Knowledge Level*. I.e. the ontology used seems to be about the KM domain, rather than addressing the software testing domain.

In order to support test case reuse, (LI; ZHANG, 2012) (#5) propose a reusable test case KM model, which is based on ontology of reusable test cases. This is a simple

ontology, indicating that a *Reusable Test Case* is a subtype of *Test Case*, and that *Reusable Test Case* has *Test Objective, Domain Area, Testing Type, Testing Method, Testing Step* and *Test Data*. *Test Data*, in turn, has *Attribute, Value,* and *Expected Result*. According to them, by using the ontology to represent the explicit specification of reusable test cases, "test engineers can find the right knowledge at the right moment for a given testing project". (LI; ZHANG, 2012) have applied the ontology representation and the KM model for software test case reuse in a third-party testing center. This testing center developed a KM system, with an organizational exchange library established on the intranet. Tacit knowledge and explicit knowledge were captured and stored in the testing knowledge warehouse, based on the analysis conducted by the knowledge analysts in the center. The reusable test case repository has more than 12,000 test cases. As a conclusion, they say that, "with the ontology and the knowledge management model, test engineers can retrieve and reuse test case flexibly and the design efficiency of test case has been improved".

Each study was analyzed in order to answer the four research questions. Following, the classification schema and data synthesis regarding each one of these questions are presented.

> ***RQ1.*** *What kinds of ontologies (considering their generality level) have been used in those initiatives?*
>
> For analyzing this question, the classification proposed by Guarino (1998) was adopted, who classifies ontologies according to their generality level into: (i) foundational (or top-level) ontologies, which describe very general concepts independently of a particular problem or domain, such as object, space, event, time, action; (ii) domain ontologies, which describe the conceptualization related to a generic domain, for example, medicine and law; (iii) task ontologies, which describe the conceptualization related to a generic task, such as diagnosis or planning; and (iv) application ontologies, which is the most specific kind of ontology, describing concepts that are dependent on a particular domain and a task. Both the ontologies presented in #1 and #5 are domain ontologies. The one presented in #1 is about the "knowledge" domain, whereas the one in #5 is about the software testing domain, more specifically, the test case domain.
>
> ***RQ2.*** *What are the purposes of using ontologies in those initiatives?*
>
> In #1, the "knowledge ontology" is used mainly for organizing the knowledge database. It classifies concepts of software testing, and describes re-

lation constraints between them. An ontology-based search and sorting mechanism is employed. When retrieving knowledge, the correlative concepts or attributes are found according to the users´ requests.

In #5, the ontology of reusable test case is designed to establish a common understanding and reduce terminological ambiguity among test engineers when reusing test cases. Moreover, it is used for representing the explicit specification of reusable test cases.

In summary, in both cases, ontologies are used for structuring the organizational memory, and for searching knowledge items.

RQ3. *What are the languages or formalisms used to create those ontologies?*

Both in #1 and #5, graphical notations are used. They do not mention that these ontologies were implemented in some coding language, such as Ontology Web Language (OWL). In #1, a specific graphical notation is used. In #5, Unified Modeling Language (UML) is used to represent the test case ontology as a class diagram, and first-order logic is used for writing a set of axioms.

RQ4. *Do the studies use the ontologies at development time, at run time or both?*

In an ontology-driven Information System (IS) in general, as well as in ontology-based KM, considering a temporal dimension, ontology can be used at development time or at run time (GUARINO, 1998). When using ontologies at development time (reference ontologies), it is important to focus on their representation adequacy, since these ontologies are used in an off-line manner to assist in tasks of meaning negotiation and consensus establishment. However, once users have already agreed on a common conceptualization, versions of a reference ontology can be created for run-time use, called operational ontologies. Operational ontologies are not focused on representation adequacy, but are designed with the focus on guaranteeing desirable computational properties (GUIZZARDI, 2007). In this context, in both cases (#1 and #5), the ontologies are used at development time, mainly for structuring the knowledge repositories.

Ontologies have been applied in a variety of areas in Computer Science (D'AQUIN; GANGEMI, 2011) and have also been widely recognized as an important technology

for implementing enterprise KM (STAAB; MAEDCHE, 2000; BENJAMINS et al., 1998; DAVIES et al., 2003; ABECKER; ELST, 2004; KIM, 2000). However, for KM in software testing, as the SLR shows, only two studies (#1 and #5) have adopted an ontology-based approach for KM in software testing. Moreover, only in #5 a software testing ontology was used. This study is quite recent (published in 2012) and seems to be a very innovative and promising research for the field of software testing. However, its focus is too narrow (test case reuse), and as a consequence the coverage of the corresponding ontology is very limited.

In both the studies (#1 and #5), ontologies are used at development time, mainly for structuring the knowledge repositories. As pointed out by Guarino (1998), an important benefit of using ontologies at development time is that ontologies enable developers to share domain knowledge using a common vocabulary. Moreover, ontologies enable developers to concentrate on the structure of the domain and the task at hand and protect them from being bothered too much by implementation details (GUARINO, 1998). In the KM context, ontologies can be used at development time for structuring knowledge repositories, as well as for establishing categories for metadata to be used for annotating knowledge items, among others. On the other hand, using ontologies at run time also presents some benefits, such as enabling the communication between software agents (GUARINO, 1998), allowing integrating structured and unstructured data, enabling inferences, and supporting semantic, metadata-based searching. Thus, it is expected that a powerful KM solution for software testing should use a software testing ontology both at development time and at run time. As a consequence, this ontology should be engineered following sound principles of Ontology Engineering.

Ontologies have been frequently employed in KM to support knowledge sharing and reuse, efficient integration of information across different applications, and to support knowledge visualization, among others (ABECKER; ELST, 2004). Ontologies are a powerful mechanism for representing knowledge and encoding its meaning (ANTUNES et al., 2007). Different research areas have benefited from the use of ontologies in KM initiatives, and software testing domain is not different. However, surprisingly, the use of ontologies in KM initiatives is not happening. And it is believed that this is a ground-breaking research that must be done. For allowing this research to happen, a well-established ontologies of the software testing domain is necessary.

2.4 Final remarks about this chapter

In this chapter, some of the most important concepts in the research areas studied were discussed. The main concepts of Software Testing and KM were described. Moreover, a study on the aspects associated with applying KM in software testing through a Systematic Mapping Study was presented.

As emphasized by Mapping Study, ontologies are an important mechanism for representing knowledge. Ontologies in KM initiatives can be a ground-breaking research. Considering this context, next chapter presents the main concepts of ontologies, as well as presents the systematic literature review on software testing ontologies.

3 ONTOLOGIES FOR SOFTWARE TESTING

This chapter describes the main concepts related to ontologies and an investigation on existing ontologies for the software testing domain. Section 3.1 presents an overview about ontologies and ontology pattern languages. Section 3.3 discusses the main ontology proposals for software testing in the literature through a Systematic Literature Review (SLR). The theoretical pillars presented in this chapter are the foundations for building a Reference Ontology on Software Testing (described in section 4.2).

3.1 Ontologies

Gruber (1993) defines ontology as an explicit specification of a conceptualization. Guarino (1998) adds that an ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations. In the simplest case, an ontology describes a hierarchy of concepts related by relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation.

Ontologies involve the description of concepts in a particular domain of knowledge with its properties and constraints. An ontology provides an accurate description of knowledge in a formal language to facilitate communication, integration, storage, search, sharing, and reusing knowledge representation (O'LEARY, 1998b). One of the main interests in ontologies is in the need for increasingly, greater interoperability and reuse of information between systems and people within an organization (RIOS, 2005). In Software Engineering, for example, ontologies have been typically used to reduce conceptual ambiguities, make transparent the knowledge structures, support knowledge sharing and interoperability between systems (USCHOLD; JASPER, 1999).

Ontology Types

There are several classifications for ontologies. In (USCHOLD et al., 1996a) and Uschold (1996b), ontologies are classified according to the degree of formality (informal, semi-informal, formal or rigorously formal). Ding and Fensel (2001) classify ontologies according to purpose. Heijst et al. (1997) classify ontologies according to

the nature and concepts structure involved. However, the classification most cited in the literature and adopted in this work is proposed by Guarino (1998) classifies ontologies according to their level of generality, as shown in Figure 3.1.

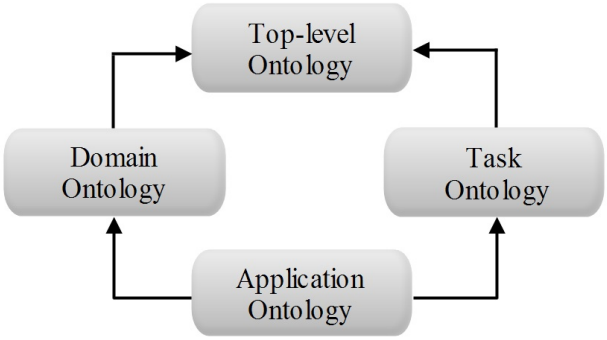

Figure 3.1 - Kinds of ontologies, according to their level of dependence on a particular task or point of view.
SOURCE: Adapted from (GUARINO, 1998)

- **Top-level Ontology:** also called Foundational Ontology, is an abstract conceptualization of generic elements, like space, time, matter, object, event, action, etc., which are independent of a particular problem or domain. The Foundational Ontology defines general concepts that underlie all other concepts. For Guizzardi (2005), Foundational Ontology are used successfully to improve the quality of modeling languages of conceptual models. Some examples are: General Formal Suggested Upper Merged Ontology (SUMO) (PEASE et al., 2002), Basic Formal Ontology (BFO)(GRENON et al., 2004), General Formal Ontology (GFO) (**??**) and Unified Foundational Ontology (UFO) (GUIZZARDI, 2005).

- **Domain Ontology:** describes the vocabulary related to a generic domain. This class represents a set of ontologies describing concepts in a domain of discourse, properties and constraints. Currently, there is a large amount of domain ontologies in many areas, such as Medicine, Law, Software Engineering, Organizational Modeling and Chemistry.

- **Task Ontology:** similar to a domain ontology, however, instead of mapping the concepts of a particular domain, maps concepts of a task, such as diagnosis, sale and registration. In other words, comprises a set of primitive task structure representation, independently of a given domain.

- **Application Ontology:** describes concepts depending both on a particular domain and task, which are often specializations of both the related ontologies. These concepts often correspond to *roles* played by domain entities while performing a certain activity, like *replaceable unit* or *spare component*.

Currently, there is a growing interest in the use of foundational ontology as conceptual modeling tools (BELLATRECHE et al., 2006). A foundational ontology describes the general categories that are used for building conceptual models.

Ontology Engineering Methods

Ontology Engineering methods have been developed in order to systematize the development of ontologies. Several methods for building ontologies have been proposed, for example, a methodology that emerged in the TOronto Virtual Enterprise (TOVE) Project (GRUNINGER; FOX, 1995), *Sensus* (SWARTOUT et al., 1996), METHONTOLOGY (GÓMEZ-PÉREZ et al., 1996) and *On-to-knoweledge* (FENSEL et al., 2000). In order to gather in a unique method some of the best features of the existing methods, Falbo and Menezes (1998b) proposed a methodology for building ontologies, called Systematic Approach for Building Ontologies (SABiO), whose main activities are show in Figure 3.2.

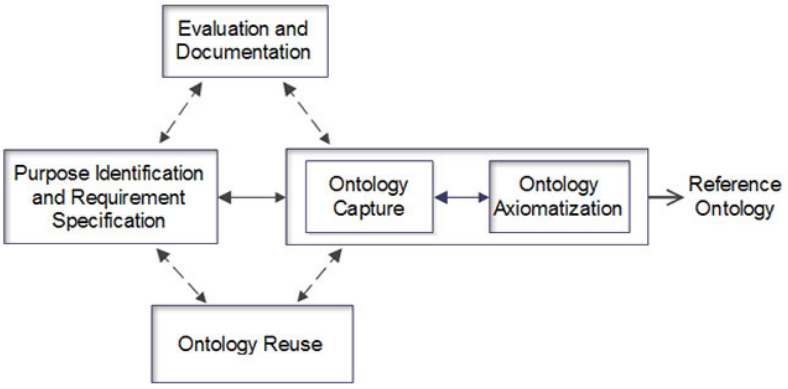

Figure 3.2 - Systematic Approach for Building Ontologies (SABiO)
SOURCE: Adapted from (FALBO; MENEZES, 1998b)

- **Purpose identification and requirement specification:** identifies the ontology purpose and its intended uses, that is, the competence of the ontology. To do that, competency questions are used.

- **Ontology capture:** captures the domain conceptualization based on the ontology competence. Relevant concepts and relations should be identified and organized. A conceptual model represented in a graphical language, and a dictionary of terms should be produced, in order to aid communication with domain experts.
- **Ontology axiomatization:** aims to explicitly represent constraints that are not captured by the conceptual model developed in the ontology capture. The axioms should be written in a formal language, such as first-order logics or Object Constraint Language (OCL).
- **Ontology Reuse:** during the mainstream phases of ontology development, it could be necessary to integrate the current ontology with existing ones, in order to reuse previously established conceptualizations.
- **Ontology evaluation:** the ontology must be evaluated to check whether it satisfies the requirements specification.
- **Documentation:** all the ontology development must be documented.

Languages for Ontology Modeling

To represent a shared conceptualisation, a language of representation is required. According to Guarino (1999), the modeling languages for ontologies have three main distinct levels (Table 3.1): logical, epistemological and ontological.

Table 3.1 - Main features of the languages for ontology modeling. Adapted from (GUARINO, 1999).

Level	**Primitive constructs**	**Main feature**	**Interpretation**	**Examples**
Logical	Predicates	Formalisation	Arbitrary	FOL
Epistemological	Structuring relations (concepts and roles)	Structure	Arbitrary	OWL, UML
Ontological	Structuring relations *satisfying meaning postulates*	Meaning	Constrained	OntoUML

Modeling languages in the logical level provides the constructs needed to formalize knowledge and allow formal reasoning, but lack cognitive foundation required to facilitate the knowledge representation in a structured way. The epistemological level languages allow not only to formalize the knowledge, but also to structure it,

favoring formal reasoning and derivation of new knowledge. However, in both levels (logical and epistemological), interpretation of the real world is completely arbitrary. In particular, at the epistemological level, despite to add a structural meaning for knowledge, it is not focused on its formal representation. On the other hand, the ontological level languages have primitives that satisfy formal meaning postulates, restricting the interpretation of a logical theory based on formal ontological distinctions.

Currently, there are several languages in the epistemological level and each has different expressivity. The most famous is the Ontology Web Language (OWL) (OWL, 2003). OWL is based on the syntax of eXtensible Markup Language (XML) (W3C, 2013) and it is developed as a vocabulary extension of Resource Description Framework (RDF) (RDF, 2004) and offers some mechanisms for semantic formalism. On the other hand, examples of ontological level languages are few. A concrete proposal in this direction is given by Guizzardi (2005), where the metamodel of class diagrams of UML is extended to incorporate the ontological distinctions defined in the Unified Foundational Ontology (UFO).

Recently, many ontologies have been developed using UML as modeling language. Although UML has deficiencies, such as inconsistency, ambiguity and lack of formal semantics to capture the basic concepts of the universe of discourse of an ontology, some works have proposed extensions and new formalisms for the UML suited to modeling ontologies (GUIZZARDI et al., 2004; EVERMANN, 2005; LI; PARSONS,). An example is OntoUML, which is designed so that it is possible to capture the semantic agreements of a universe of discourse and view them in the form of class diagrams of UML. OntoUML is a UML profile that enables modelers to make finer-grained modeling distinctions between different types of classes and relations according to some ontological distinctions put forth by UFO (GUIZZARDI, 2005).

Ontology Patterns

Although there is a wide range of ontology engineering methods, tools and languages, building ontologies is still a difficult task. In this context, reuse is pointed out as a promising approach for ontology engineering. Ontology reuse allows speeding up the ontology development process. Ontology Patterns (OP) are an emerging approach that favors the reuse of experiences and good practices. According to Falbo et al. (2013), an OP describes a particular recurring modeling problem that arises in specific ontology development context and presents a well-proven solution for this problem. The next section presents a brief description of Ontology Pattern

Languages.

In summary, ontology patterns are considered a promising approach that favors reuse of encoded experiences and good practices in Ontology Engineering (PRESUTTI et al., 2009). Moreover, core ontologies organized as Ontology Pattern Language (OPL) have potential to amplify the benefits of ontology patterns (FALBO et al., 2013).

3.2 Ontology Pattern Languages

An OPL aims to provide holistic support for using Domain-related Ontology Patterns (DROPs) in ontology development for a specific application domain. It provides explicit guidance on what problems can arise in that domain, informs the order to address these problems, and suggests one or more patterns to solve each specific problem. An OPL gives concrete and thoughtful guidance for developing ontologies in a given domain (FALBO et al., 2013).

Falbo et al. (2013) presented an OPL on Software Process. Its patterns were extracted from the Software Process Ontology (SPO) presented in (BRINGUENTE et al., 2011). SPO was restructured in a Software Process Ontology Pattern Language (SP-OPL). As an ontology pattern language, SP-OPL contains a set of interrelated ontology patterns related to the software process domain. SP-OPL provides a precise definition of the structural knowledge in the field of software processes that spans across different application domains in this field (FALBO et al., 2013). Moreover, SP-OPL is grounded on the UFO.

Figure 3.3 shows the SP-OPL patterns and Table 3.2 shows the patterns that compose the SP-OPL. Details of some of the patterns are shown in more detail in Chapter 4 and Annex A.

Table 3.2 - Domain-Related Ontology Patterns (DROPs) in the SP-OLP (FALBO et al., 2013)

Id	Name	Intent
Standard Process Definition		
SPS	Standard Process Structure	Represents how a standard software process is defined in terms of standard sub-processes and activities
HRD	Standard Activity Human Role Definition	Defines the human roles responsible for performing a standard activity in the projects that instantiate it
RTD	Standard Activity Resource Type Definition	Defines the types of resources (hardware and software) required for performing a standard activity

Continues

Table 3.2 - Conclusion

Id	Name	Intent
WPD	Standard Activity Work Product Definition	Defines the types of work products required (input) and produced (output) when performing a standard activity
PD	Standard Activity Procedure Definition	Defines the procedures (methods, techniques, guidelines etc.) to be applied when performing a standard activity
Project Process Definition		
PP	Process Planning	Represents how a process is planned in terms of sub-processes and activities
HRP	Human Role Planning	Defines the human roles responsible for performing a project activity
RP	Resource Planning	Defines the types of resources (hardware and software) required for performing a project activity
WPP	Work Product Planning	Defines the types of work products required (input) and produced (output) when performing a project activity
PRP	Procedure Planning	Defines the procedures (methods, techniques, guidelines etc.) to be applied when performing a project activity
Resource Allocation		
PTD	Project Team Definition	Defines the human resources that are member of a project team
TDHRA	Team-dependent Human Resource Allocation	Allocates human resources to project activities, considering team allocation constraints
TIHRA	Team-independent Human Resource Allocation	Allocates human resources to project activities, when there is not a project team formally defined
RAL	Resource Allocation	Allocates resources (hardware equipments and software tools) to project activities
Process Scheduling		
PSCH	Process Scheduling	Defines the time boundary for project processes and activities
Software Process Execution		
PAE	Process and Activity Execution	Register the occurrences of processes and activities
PAET	Process and Activity Execution and Tracking	Register the occurrences of processes and activities, taking previous scheduled processes and activities into account
HRP	Human Resource Participation	Registers the participation of a Human Resource in an activity occurrence
RPA	Resource Participation	Registers the participation of a Resource (hardware equipment or software tool) in an activity occurrence

Now, in order to address aspects such as Organization Arrangement, Definition Team, Institutional Roles, Institutional Goals, and Human Resource Management, the Enterprise-Ontology Pattern Language (E-OPL) was developed (FALBO et al., 2014). Its first version addresses five aspects common to several enterprises, as shown

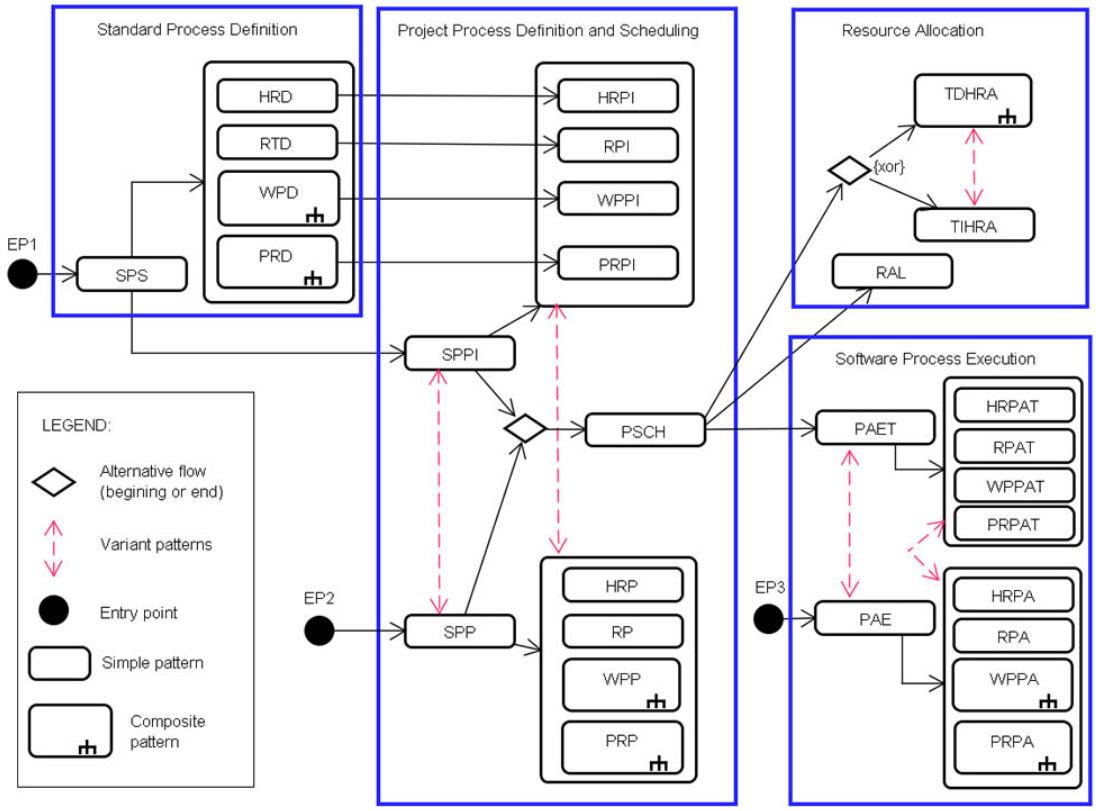

Figure 3.3 - Software Process Ontology Pattern Language (SP-OPL)
SOURCE: (FALBO et al., 2013)

in Figure 3.4.

1. **Organization Arrangement**: includes patterns related to how an organization is structured in terms of organizational units, as well as how complex organizations are organized in terms of other organizations.

2. **Team Definition**: deals with defining teams for projects, organizations or organizational units.

3. **Institutional Roles**: focuses on roles and positions to be played by enterprise employees.

4. **Institutional Goals**: deals with institutional agents´ goals.

5. **Human Resource Management**: treats several human resource relations in an enterprise, such as employment, allotment to an organizational unit, team allocation, and position occupation.

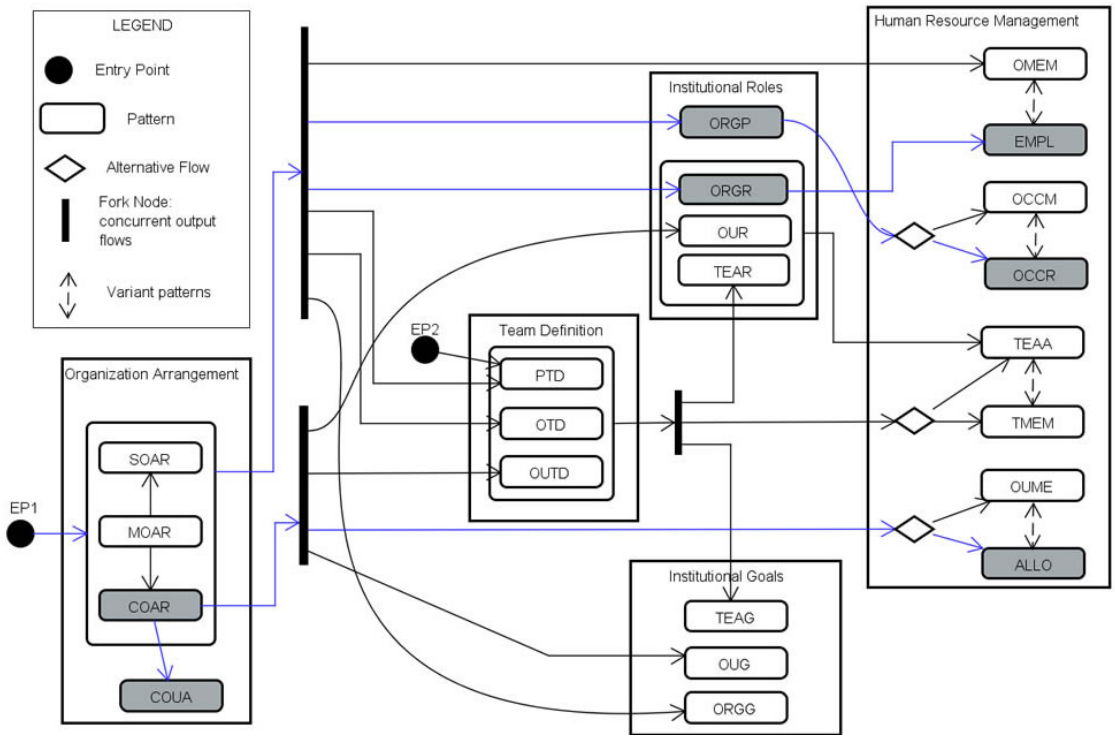

Figure 3.4 - Enterprise Ontology Pattern Language (E-OPL)
SOURCE: (FALBO et al., 2014)

The patterns defined in SP-OPL and E-OPL are the essential elements in building the Reference Ontology on Software Testing, presented in Chapter 4.

3.3 Software Testing Ontology: Systematic Literature Review

Ontologies are recognized as an important instrument by the KM. Since studies have actually used an ontology-based approach for KM, efforts in such research area showed to be promising. Thus, looking for a domain ontology that can be used in a KM initiative in software testing, this section presents an investigation of ontologies in the software testing domain which addresses the second goal of this thesis: conduct a SLR with purpose of investigating how widespread are ontologies for software testing, and the characteristics of such ontologies.

3.3.1 Review Protocol

A protocol (SOUZA et al., 2013c) for the entire process of the SLR was developed and is presented below (Table 3.3):

Research Questions: This SLR aims at answering the following research ques-

tions:

Table 3.3 - Research questions and their rationales

N^o	**Research question**	**Rationale**
RQ1	What is the coverage of the software testing domain in the existing ontologies?	In this research question is necessary to know the coverage of the software testing domain, that is, which activities of the testing process are covered.
RQ2	How were they developed?	This question investigates the main way that the ontologies were engineered, such as references and international standards considered, engineered method, languages used and foundational ontologies.

Inclusion and Exclusion Criteria: The selection criteria are organized in one inclusion criterion (IC) and five exclusion criteria (EC). The inclusion criterion is: (IC1) The study presents an ontology about the software testing domain. The exclusion criteria are: (EC1) The study does not have an abstract; (EC2) The study is just published as an abstract; (EC3) The study is not written in English; (EC4) The study is an older version (outdated) of another study already considered; and (EC5) The study is not a primary study, such as editorials, summaries of keynotes, workshops, and tutorials.

Sources: Search was done in eight electronic databases that were considered the most relevant according to (DYBA et al., 2007). They are:

IEEE Xplore (*http://ieeexplore.ieee.org*)
ACM Digital Library (*http://dl.acm.org*)
SpringerLink (*http://www.springerlink.com*)
Scopus (*http://www.scopus.com*)
Science Direct (*http://www.sciencedirect.com*)
Compendex (*http://www.engineeringvillage2.org*)
ISI of Knowledge (*http://www.isiknowledge.com*)
DBLP Computer Science Bibliography (*http://dblp.uni-trier.de/*)

Keywords and Search String: The search string considers two areas: Software

Testing and Ontology (Table 3.4), and it was applied in three metadata fields (title, abstract and keywords). The search went through syntactic adaptations according to particularities of each source.

Table 3.4 - Keywords of the Search String of the SLR.

Areas	**Keywords**
Software Testing	"Software Testing", "Software Test"
Ontology	"ontology", "ontologies"
Search String: ("Software Testing" OR "Software Test") AND ("Ontology" OR "Ontologies")	

Assessment: Before conducting the SLR the review protocol was tested. This test was conducted in order to verify its feasibility and adequacy, based on a pre-selected set of studies considered relevant to investigation. The review process was conducted just this author of this thesis and the other two carried out its validation. Approximately 35% of the studies were analyzed using two different samples.

3.3.2 Conducting the Review

Using the search string, 396 records were retrieved. The selection process applied on the returned publications was performed in three stages. In the first stage, duplicates were eliminated by examining title and abstract, since several publications are available in more than one source. In the second stage, inclusion and exclusion criteria were applied considering also title and abstract. Finally, in the third stage, the exclusion criteria were applied considering the entire text. After applying the selection criteria, 18 studies remained. Table 3.5 shows the progressive reduction of the number of studies throughout the selection process for the review.

A selection process comprising three stages were followed, as shown in Figure 3.5. From the 18 studies, 12 different ontologies were identified. This difference comes from the fact that some papers present different parts or evolutions of the same ontology. As the result of this SLR, the following testing ontologies were found: Software Testing Ontology for Web Service (STOWS) (HUO et al., 2003; ZHU; HUO, 2005; HONG, 2006; YUFENG; HONG, 2008; ZHU; ZHANG, 2012), OntoTest (BARBOSA et al., 2006; NAKAGAWA et al., 2009), TaaS Ontology (YU et al., 2008; YU et al., 2009), and the ontologies proposed in (LI; ZHANG, 2012),(ARNICANS et al., 2013), (GUO et al., 2011), (NASSER et al., 2009), (BAI et al., 2008), (RYU et al., 2011), (SAPNA; MOHANTY,

Table 3.5 - Result of the Selection Process Stages of the SLR.

Stage	Criteria	Analyzed Content	Inicial Studies	Final Studies	Reduction (%)
1	Eliminating duplication	Title and abstract	396	295	25,5%
2	IC1, EC1, EC2, EC3, EC4, EC5	Title and abstract	295	30	89,8%
3	IC1, EC4, EC5, EC6	Entire Text	30	18	40%

2011), (CAI et al., 2009) and (ANANDARAJ et al., 2011).

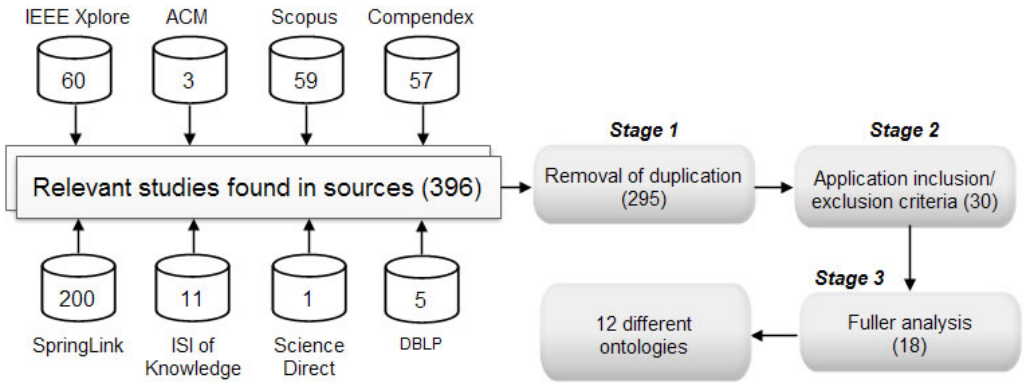

Figure 3.5 - Search and selection SLR process

From the 12 identified ontologies, extensions, evolutions and/or other publications that present the ontologies more completely were analyzed. It was the case of OntoTest. OntoTest has a testing resource sub-ontology presented in (BARBOSA et al., 2008). However, this study did not return in the SLR, probably because the searched sources do not contain this paper or because they failed to identify it by the search string.

3.3.3 Review Results

After selecting the primary studies, each one was analyzed in order to answer the research questions presented in section 3.3.1. Next, the data synthesis regarding these questions is presented.

RQ1. *What is the coverage of the software testing domain in the existing ontologies about this domain?*

Regarding domain coverage, it was noticed that most ontologies have very limited coverage. The ontology presented in Guo et al. (2011) specifies only the concept of test case. The ontology presented in Li and Zhang (2012) focuses also on test case, but considering some concepts related to test process. The ontology presented in Bai et al. (2008), called Test Ontology Model (TOM)), models only testing artifacts and the relationships between them. The ontologies presented in Arnicans et al. (2013), Cai et al. (2009) and Anandaraj et al. (2011) are, in fact, taxonomies. These ontologies only present a simple structure of the domain concepts of software testing, and thus they do not qualify as ontologies, or, at most, they are lightweight ontologies.

The ontology presented in Nasser et al. (2009) is devoted to state machine based testing. The ontology presented in Sapna and Mohanty (2011), focus on scenario-based testing, though it captures general testing concepts too. The ontology presented in Ryu et al. (2011) is not properly a testing ontology, but it is an OWL implementation of a specific testing maturity model developed by the authors (the Ministry of National Defense-Testing Maturity Model (MND-TMM)).

The ontologies that have higher coverage are: STOWS (HUO et al., 2003; ZHU; HUO, 2005; HONG, 2006; YUFENG; HONG, 2008; ZHU; ZHANG, 2012), OntoTest (BARBOSA et al., 2006; BARBOSA et al., 2008; NAKAGAWA et al., 2009), TaaS Ontology (YU et al., 2008; YU et al., 2009).

STOWS classifies its concepts into three categories: (i) elementary concepts, which are general concepts about computer software and hardware; (ii) basic testing concepts, which include the concepts of Tester, Artifact, Activity, Context, Method, and Environment; and (iii) compound testing concepts, which combine basic testing concepts, giving rise to the concepts of Task and Capability. STOWS presents a set of taxonomies of each basic testing concept, including also some properties and few relations.

The TaaS Ontology has two core concepts (Test Task and Test Capability), which are composite concepts aggregating other concepts. Test Task consists of Test Activity, Test Type, Target Under Test, Test Environment, and Test Schedule. Test Capability, in turn, consists of Test Type, Test Activity, Test Environment, Target Under Test and Quality of Service.

Finally, OntoTest is a modular ontology, built in layers. OntoTest is composed by a "Main Software Testing Ontology", and six sub-ontologies (BARBOSA et al., 2006): Testing Process, Testing Phase, Testing Artifact, Testing Step, Testing Resource, and Testing Procedure sub-ontologies. The Main Software Testing Ontology is presented in (BARBOSA et al., 2006). It is a simple model that includes six concepts. According to this model, a Testing Process is composed by Testing Steps, and it has Testing Phases. A Testing Step requires Testing Resources, adopts Testing Procedures, consumes and generates Testing Artifacts, and depends on other Testing Steps. Testing Artifacts can depend on other Testing Artifacts, and can be composed by other Testing Artifacts. Finally, a Testing Procedure can be supported by Testing Resources, and is adequate to Testing Process.

OntoTest Testing Step sub-ontology introduces the concept of Testing Activity, indicating that a Testing Step is composed by Testing Activities, while Testing Activities are not further decomposed. The remainder of this sub-ontology consists of two large taxonomies: a Testing Step taxonomy, and a Testing Activity taxonomy. The Testing Resource sub-ontology (BARBOSA et al., 2008) has a taxonomy of types of resources. This taxonomy is organized in two branches: Human Resources (which can be members of Test Teams), and Testing Environment, which is further extended in Software and Hardware Resource. Software Resource is further extended into Testing Tool and Supporting System. Testing Tool can be composed by several types of Testing Modules. No papers were found that presenting the Testing Process, Testing Phase, Testing Artifact, and Testing Procedure sub-ontologies. So, it is believed that OntoTest is a work in progress.

RQ2. *How were the testing ontologies developed?*

Concerning this research question, some aspects related to the way the ontologies were engineered was considered, namely:

(i) Do the ontologies try to capture a common (shared) conceptualization of the testing domain, taking into account different references and especially international standards?

(ii) Are the ontologies developed following an ontology engineering method (including some sort of evaluation)?

(iii) In which abstraction level (conceptual and implementation levels) are the ontologies developed? Which are the languages used?

(iv) Do the ontologies take foundational aspects (foundational ontologies) into account?

The first aspect investigated is if the ontologies try to capture a common conceptualization of the testing domain. Some ontologies take international standards into account: OntoTest is based on 1st edition of ISO/IEC 12207 (ISO/IEC, 2008); the ontology presented in Arnicans et al. (2013) was created based on the glossary "Standard glossary of terms used in Software Testing" of the International Software Testing Qualifications Board-ISTQB; the ontology presented in Bai et al. (2008) is based on the Unified Modeling Language 2.0 Test Profile (U2TP); and the ontologies presented in Cai et al. (2009) and Sapna and Mohanty (2011) are based on the SWEBOK (ABRAN et al., 2004). The other studies neither mention the use of international standards as basis for their ontologies, nor which references were used as basis for developing the ontologies. The exception is the ontology presented in (RYU et al., 2011), which, as said before, is an OWL implementation of the MND-TMM. It is worthwhile to point out that, despite some ontologies are based on international standards, generally they take only one standard into account, and thus they do not consider a broad set of testing references to really establish a common (consensual) conceptualization.

Regarding the methods adopted for building the ontologies, Arnicans et al. (2013) propose a method for semi-automatic obtaining lightweight ontologies, which uses the ONTO6 method. In Sapna and Mohanty (2011) ideas from two methods for building ontologies: METHONTOLOGY (JURISTO et al., 1997) and Ontology Development 101 (NOY; MCGUINNESS, 2001). Cai et al. (2009) used the skeletal method (USCHOLD; KING, 1995) for building their testing ontology. OntoTest was built using a method that combines guidelines given by SABiO and METHONTOLOGY, with focus on ontology capture and formalization. Finally, in Anandaraj et al. (2011) a very simple method was followed, comprising four steps, namely: (i) determine domain and scope of the ontology; (ii) define concepts in the ontology; (iii) create a class hierarchy; and (iv) define properties and constraints. The other studies do not mention if a method (or which method) was used for building the proposed ontologies.

Although the aforementioned ontologies have been developed following methods that include activities devoted to ontology evaluation, such as Skeletal method, SABiO and METHONTOLOGY, none of the studies discussed how the ontologies were evaluated, except Arnicans et al. (2013), which says that a software testing expert

has analyzed the ontology fragment related to testing techniques.

Regarding the abstraction level, 7 of the 12 studies (58.3%) present their ontologies as conceptual models, namely: STOWS, OntoTest, TaaS Ontology and the ontologies presented in Li and Zhang (2012), Arnicans et al. (2013), Bai et al. (2008) and Sapna and Mohanty (2011). 5 of the 12 studies (41.7%) present the ontologies only as a code artifact (implemented in OWL), namely: the ontologies presented in Guo et al. (2011), Nasser et al. (2009), Ryu et al. (2011), Cai et al. (2009) and Anandaraj et al. (2011). The following ontologies are represented in both conceptual and implementation levels: STOWS, OntoTest, and the ontologies presented in Arnicans et al. (2013), Bai et al. (2008) and Sapna and Mohanty (2011). It is important to clarify the approach followed in Arnicans et al. (2013). In this study, first the ontology is semi-automatically generated in OWL. The obtained ontology is then transformed to UML class diagram using a tool called OWLGrEd in order to be evaluated by experts.

Concerning the languages used for representing the ontologies, all the studies that present the ontologies in the implementation level used OWL. In the conceptual level, all the ontologies are presented as UML class diagrams. Moreover, two ontologies use first order logic to capture some axioms, namely OntoTest and the ontology presented in Li and Zhang (2012).

Summarizing, from the 12 ontologies investigated, 2 are represented only as conceptual models presented as UML class diagrams (TaaS Ontology, and the ontology presented in Li and Zhang (2012)), 5 are represented only as OWL implementations (the ontologies presented in Guo et al. (2011), Nasser et al. (2009), Ryu et al. (2011), Cai et al. (2009) and Anandaraj et al. (2011)), and 5 are represented both in the conceptual level (as UML class diagrams) and in the implementation level (as OWL artifacts) (STOWS, OntoTest, and the ontologies presented in Arnicans et al. (2013), Bai et al. (2008), and Sapna and Mohanty (2011)).

Finally, although foundational ontologies have been recognized as an important instrument for improving the quality of conceptual models in general, and more specifically of domain ontologies (GUIZZARDI, 2007), none of ontologies analyzed in our SLR reuses foundational ontologies.

3.3.4 Discussion

Currently, software testing is considered a complex process comprising activities, techniques, artifacts, and different types of resources (hardware, software and human resources). Thus, building a complete testing ontology is not a trivial task (if even possible).

Although there are a relatively large number of ontologies on software testing published in the literature (12 ontologies), there are still problems related to the establishment of an explicit common conceptualization regarding this domain. For being applied to KM, a software testing ontology must take some characteristics of good quality ontologies into account.

In an experiment trying mainly to identify good practices in ontology design, D'Aquin and Gangemi (2011) have identified some characteristics that are presented in what they call "beautiful ontologies". These characteristics were grouped in three dimensions: (i) formal structure, (ii) conceptual coverage and task, and (iii) pragmatic or social sustainability. In order to evaluate the testing ontologies selected by means of the SLR, only the first dimension and in part of the second were focused, namely conceptual coverage. The characteristics included in these dimensions are (D'AQUIN; GANGEMI, 2011):

> **Structure:** the ontology reuses foundational ontologies; the ontology is designed in a principled way; it is formally rigorous; it also implements non-taxonomic relations; the ontology strictly follows an evaluation method; it is modular, or embedded in a modular framework.
>
> **Conceptual coverage:** the ontology provides important reusable distinctions; it has a good domain coverage; it implements an international standard; the ontology provides an organization to unstructured or poorly structured domains.

Unfortunately, some of these characteristics are difficult to evaluate, since there isn't much information about them in the papers presenting the corresponding ontologies. Thus, in this analysis, the most easily discernible features was focused, namely: having a good domain coverage; implementing an international standard; being formally rigorous; implementing also non-taxonomic relations; following an evaluation method; and reusing foundational ontologies. Table 3.6 below shows the comparison of the 12 ontologies found according to the characteristics analyzed.

Regarding the first characteristic (having a good domain coverage), as shown in Table 3.6, most ontologies have very limited coverage. The ontologies that have higher coverage are: STOWS, OntoTest, and TaaS.

Some of the ontologies take international standards into account, namely: OntoTest, and the ontologies presented in Arnicans et al. (2013), Bai et al. (2008), Sapna and Mohanty (2011), and Cai et al. (2009). Others, on the other hand, do not consider international standards (or at least do not mention them). This is the case of STOWS and TaaS Ontology.

The next two characteristics (being formally rigorous and also implementing non-taxonomic relations) are very important for a reference ontology. As discussed previously, a reference ontology must be a heavyweight ontology, and thus it must comprise conceptual models that include concepts, and relations (of several natures), and also axioms describing constraints and allowing to derive information from the domain models. Taking this perspective into account, most of the existing ontologies present problems.

There are five ontologies (the ontologies presented in Guo et al. (2011), Nasser et al. (2009), Ryu et al. (2011), Cai et al. (2009) and Anandaraj et al. (2011)) that are just OWL artifacts (i.e., operational ontologies), and thus are not enough for the purposes of applying ontologies for KM.

Table 3.6 - Comparison of the ontologies for software testing

Ontology	Coverage	References	Engineering Method	Evaluation Method	Abstraction Level	Languages	Foundational Ontologies
STOWS	Tester, Artifact, Activity, Context, Method, Environment, Task and Capability. Is mainly a set of taxonomies of basic concepts, including some properties and few relations	-	-	-	conceptual models, implementation	UML, OWL	-
OntoTest	In 6 subontologies: Testing Process, Testing Phase, Testing Artifact, Testing Step, Testing Resource, and Testing Procedure sub-ontologies. Desenvolvidas apenas 2	ISO/IEC 12207	SABiO, METHONTOLOGY	-	conceptual models, implementation	UML, OWL, axioms	-
Taas	Test Task (Test Activity, Test Type, Target Under Test, Test Environment, and Test Schedule) and Test Capability (Test Type, Test Activity, Test Environment, Target Under Test and Quality of Service)	-	-	-	conceptual models	UML	-
Li and Zhang (2012)	Test Case	-	-	-	conceptual models	UML, axioms	-
Arnicans et al. (2013)	Simple taxonomies	ISTQB	ONTO6 method	Software testing expert has analyzed the ontology	conceptual models, implementation	UML, OWL	-
Guo et al. (2011)	Test Case	-	-	-	Implementation	OWL	-

Continues

Table 3.6 - Conclusion

Ontology	**Coverage**	**References**	**Engineering Method**	**Evaluation Method**	**Abstraction Level**	**Languages**	**Foundational Ontologies**
Bai et al. (2008)	Testing Artifact	U2TP	-	-	conceptual models, implementation	UML, OWL	-
Ryu et al. (2011)	TMM	-	-	-	Implementation	OWL	-
Sapna and Mohanty (2011)	General testing concepts	SWEBOK	METHONTOLOGY, Development 101	-	conceptual models, Implementation	UML, OWL	-
Cai et al. (2009)	Simple taxonomies	SWEBOK	-	-	implementation	OWL	-
Anandaraj et al. (2011)	Simple taxonomies	-	Very simple method was followed	-	implementation	OWL	-
Nasser et al. (2009)	Devoted to state machine based testing	-	-	-	implementation	OWL	-

The ontologies presented in Arnicans et al. (2013) and Cai et al. (2009) are, in fact, taxonomies, and thus, they do not qualify as ontologies (or at most, they are lightweight ontologies). STOWS is mainly a set of taxonomies of basic concepts, including some properties and few relations. There are taxonomies of *Tester*, *Context*, *Testing Activities*, *Testing Methods*, and *Testing Artifacts*, but there are important relations missing. For instance, which are the artifacts produced and required by a testing activity? Without relations between the concepts, questions such as this one cannot be answered. Moreover, there are two "compound concepts" in STOWS that are defined on the bases of the basic concepts: capability and task. Capability, for instance, is modeled as a composite entity, which parts are *Activity*, *Method*, an optionally *Environment*, *Context*, and *Data* (a subtype of Artifact). This model is questionable, since it puts together objects and events as part of *Capability*. Objects (or endurants) exist in time; while events (or perdurants) happen in time (GUIZZARDI et al., 2008). So what is a Capability? An object or an event? This shows that this ontology presents problems.

TaaS Ontology presents very simple models. The UML class diagrams presented in Yu et al. (2008) and Yu et al. (2009) do not specify multiplicities of the relationships. Moreover, like STOWS, most relationships are modeled as aggregations (whole-part relations in UML). This approach is very questionable from an ontological point of view. For instance, there is a core concept called Test Task, which is modeled as composed by *TestActivity*, *TestType*, *TargetUnderTest*, *TestEnvironment* and *TestSchedule*. Analogously to the analysis on STOWS, the composite object Test Task aggregates endurants and perdurants.

Although probably the most complete ontology among the ones achieved through the SLR, OntoTest also presents problems. First, there are sub-ontologies that were not published yet, namely the *Testing Process, Testing Phase, Testing Artifact*, and *Testing Procedur*e sub-ontologies. Second, OntoTest does not properly link the concepts in the sub-ontologies. For instance, albeit in the *Main Software Testing Ontology* there is a relationship between Testing Step and *Test Resource*, there aren't relationships between their subtypes. This is an important part of the software testing conceptualization that needs to be made explicit.

Regarding ontology evaluation, none of the works in the SLR discusses how the ontologies they propose were evaluated. Except Arnicans et al. (2013), which says that a software testing expert has analyzed the ontology fragment related to testing techniques.

Finally, concerning the reuse of foundational ontologies, none of the ontologies analyzed have used one. This can be considered a problem, because important distinctions made in Formal Ontologies may be disregarded as clearly noticed in the brief analysis done (as in the aforementioned cases of STOWS and TaaS Ontology). The lack of truly ontological foundations puts in check the truthfulness of those ontologies.

Thus, as the main finding of SLR, concludes that the software testing community has still a lot work to do, in order to advance towards a reference software testing ontology. Once developed a good quality reference testing ontology, an operational version of it should be designed and implemented. Based on the results of the SLR, an ontology of software testing has been developed as part of the objectives of this research work and is presented in section 4.2.

3.4 Final remarks about this chapter

In this chapter the ontology theme and related concepts were introduced. The main proposals of ontologies for software testing were also discussed through a SLR. As a result, 12 ontologies were found. However, analyzing them, it was observed that they were inappropriate for purposes of this thesis. None of the ontologies is considered a *de facto* standard for the area, neither covers aspects of the entire software testing process. Moreover, the proposed ontologies for software testing in the literature have several limitations, especially with respect to coverage and validation.

The theoretical pillars presented in this chapter were the foundation for building a Reference Ontology on Software Testing (ROoST). ROoST is a reference domain ontology, that is, a domain ontology that was constructed with the main goal of making the best possible description of the testing domain. In (SOUZA et al., 2013b) there is a first version of ROoST.

Considering the results and analysis performed by systematic mapping of KM in software testing (Section 2.3.1) and SLR on ontologies for software testing (Section 3.3), the next chapter presents an ontology-based framework for guiding KM initiatives in the software testing domain that consists of a reference ontology for the software testing domain and a process with a set of directions for implementing a KMS in software testing organizations.

4 AN ONTOLOGY-BASED FRAMEWORK FOR KNOWLEDGE MANAGEMENT IN SOFTWARE TESTING

Although the importance of Knowledge Management (KM) has been widely promoted and recognized in several areas, few organizations are truly capable of performing and managing knowledge in their organizations. This is because implementing KM is not an easy task that organizations can undertake (WONG; ASPINWALL, 2004). According to Storey and Barnett (2000), a large number of organizations are taking great interest in the idea of KM and many are launching KM initiatives, but a large part of such initiatives fails. This is because, the organizations do not know how and where to start and lack the guidance of a proper and cohesive implementation framework.

According to the Wong and Aspinwall (2004), a framework is a basic structure underlying a system. A framework is a set of basic assumptions or fundamental principles of intellectual origin that forms the underlying basis for action. Thus, it can be interpreted as a structure that comprises relevant entities or a set of guiding principles and ideas that support a discipline. If KM is to be accomplished, a structure, a set of principles or a framework is needed to underpin and provide a theoretical basis for performing the relevant actions and activities (WONG; ASPINWALL, 2004). Rubenstein-Montano et al. (2000) claim that KM frameworks provide guidance and direction on how KM should be carried out.

This chapter presents Testing-Knowledge Management (T-KM), an ontology-based framework for guiding KM initiatives in the software testing domain, supported by Knowledge Management System (KMS). T-KM consists of a reference ontology for the software testing domain and a process with a set of guidelines for implementing a KMS in software testing organizations. Section 4.1 presents an overview of T-KM. Section 4.2 presents ROoST, a *Reference Ontology on Software Testing*. Section 4.3 presents the process for applying KM in software testing.

4.1 Framework Overview

Figure 4.1 presents the main components of T-KM framework, and their relationships with the resulting KMS for supporting KM in software testing.

ROoST component is a reference domain ontology, i.e. a domain ontology that was constructed with the main goal of making the best possible description of the testing domain. ROoST was developed for establishing a common conceptualization about

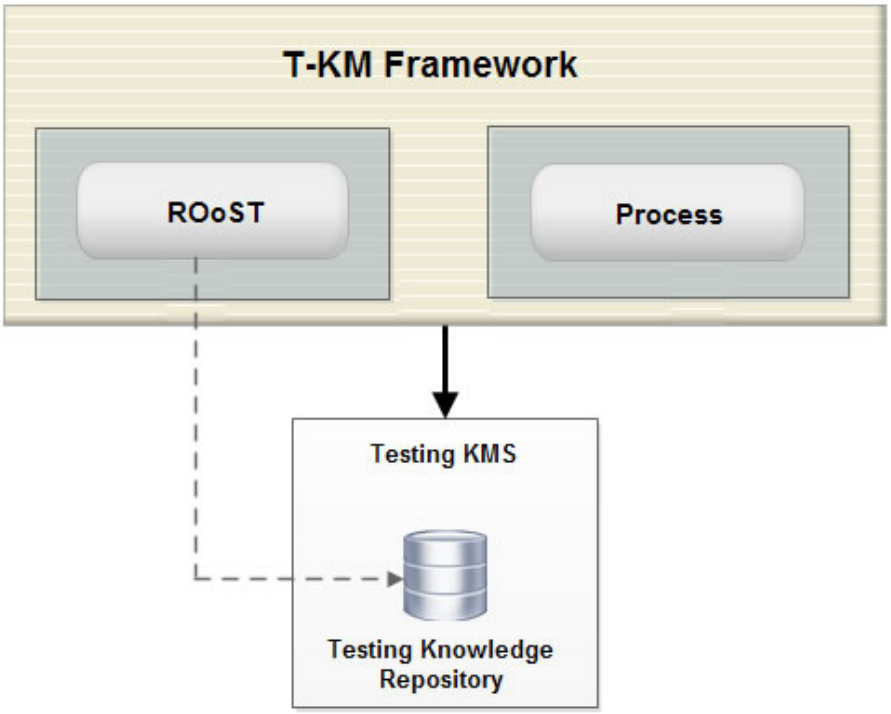

Figure 4.1 - T-KM Framework components

the software testing domain, and can be used to serve several KM-related purposes, such as defining a common vocabulary for knowledge workers regarding the testing domain, structuring knowledge repositories, annotating knowledge items, and for making searching easier. In particular, ROoST can be used as a sketch of the conceptual model for defining the structure of the Testing Knowledge Repository, as shown by the dashed line linking ROoST to the Testing Knowledge Repository.

The T-KM Process component provides a series of steps that are guidelines to be followed in the implementation of KM initiatives in software testing supported by a KMS. The proposed steps are: (i) Diagnose the Current State of the Organization's Testing Process; (ii) Establish the Scope of the Testing KM Initiative; (iii) Develop a Testing KMS; (iv) Load Existing Knowledge Items; and (v) Evaluate the Testing KMS.

The following sections describe in detail each T-KM component Framework.

4.2 Reference Ontology on Software Testing (ROoST)

This section presents the *Reference Ontology on Software Testing* (ROoST), the first component of framework for conducting KM initiatives in software testing and which

addresses the third goal of this thesis. Section 4.2.1 presents the approach adopted for building ROoST. Section 4.2 presents ROoST itself. Section 4.2.6 discusses ROoST evaluation.

4.2.1 Ontology Engineering Approach

In order to develop ROoST, method Systematic Approach for Building Ontologies (SABiO) (FALBO, 2004a) was adopted. SABiO prescribes an iterative process comprising the following activities: purpose identification and requirement specification, ontology capture, ontology formalization, reuse of existing ontologies, ontology evaluation, and ontology documentation (see Section 3.1).

The purpose of ROoST is to define a shared vocabulary regarding the testing domain to be used in KM initiatives to facilitate communication, integration, search, and representation of testing knowledge. In order to achieve this purpose, ROoST should be able to answer the following competency questions:

CQ01. What is the project in which a given testing process occurred?

CQ02. How is a testing process structured in terms of testing activities and sub-activities?

CQ03. When did a testing process start and when did it end?

CQ04. When did a testing activity start and when did it end?

CQ05. On which activities does a testing activity depend on to be performed?

CQ06. What are the test levels typically considered in testing?

CQ07. What are the artifacts produced in a testing activity?

CQ08. What are the artifacts used by a testing activity?

CQ09. How do testing artifacts relate to each other?

CQ10. Which are the testing techniques adopted in a testing activity devoted to designing test cases?

CQ11. To which test levels a testing technique can be applied?

CQ12. Which are the testing techniques applied to derive a given test case?

CQ13. Which human resources participate in a testing activity?

CQ14. When did a human resource participation start and when did it end?

CQ15. Which hardware resources are used in a testing activity?

CQ16. When did a hardware resource participation start and when did end?

CQ17. Which software resources are used in a testing activity?

CQ18. When did a software resource participation start and when did it end?

CQ19. Which are the hardware, software, and human resources that comprise the testing environment of a project?

During ontology capture, the use of a graphical language is essential in order to facilitate the communication between ontology engineers and experts. The conceptual models of ROoST are encoded in OntoUML (GUIZZARDI, 2005) (presented in section 3.1).

For developing ROoST, several references were used, including international standards. The main literature references used for building ROoST were: (IEEE, 1990; IEEE, 1998; MYERS, 2004; ABRAN et al., 2004; PRESSMAN, 2006; BLACK; MITCHELL, 2008; MATHUR, 2012).

With respect to ontology reuse, patterns from Software Process Ontology Pattern Language (SP-OPL) (FALBO et al., 2013) were reused. As presented in section 3.2, SP-OPL is a core ontology on software processes it and is grounded on the UFO (GUIZZARDI et al., 2008). Although the main concepts in ROoST have a counterpart in SP-OPL, when extending SP-OPL conceptualization for the testing domain, new concepts that were not described in SP-OPL were also introduced. In those cases, in order to maintain the alignment with Unified Foundational Ontology (UFO), concepts introduced in ROoST were also analyzed in the light of UFO. Moreover, patterns from Enterprise Ontology Pattern Language (E-OPL) (FALBO et al., 2014) also were used.

For developing ROoST, the third entry point (EP3) of SP-OPL was chosen, since the interest is to represent knowledge involved in the execution of testing processes. Figure 4.2 shows the SP-OPL patterns accessible from this entry point. Patterns Human Resource Participation (HRPA), Resource Participation (RPA), Work Product Participation (WPPA) and Procedure Participation (PRPA) were reused. The pattern used in the development of ROoST are presented in details in Annex A.

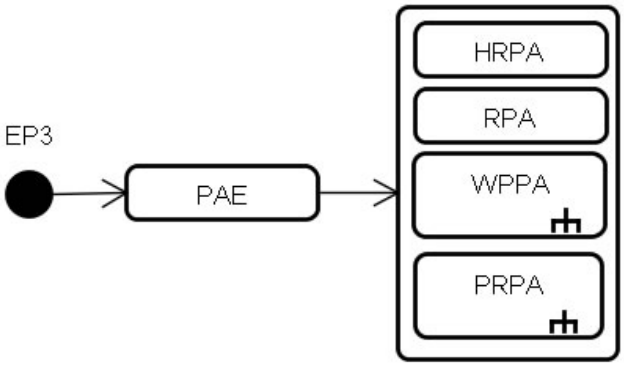

Figure 4.2 - SP-OPL patterns accessible from the entry point EP3.
SOURCE: Falbo et al. (2013)

It is important to emphasize that SP-OPL drove the rewriting of the competency questions originally defined for ROoST. Once reused patterns of SP-OPL, competency questions were reused and adapted to the software testing domain. However, very specific questions about the software testing domain that do not have a counterpart in SP-OPL were also considered. This is the case of *CQ06*, *CQ09*, *CQ12* and *CQ19*.

ROoST is developed in a modular way. Currently, ROoST has four modules (sub-ontologies). In Figure 4.3 sub-ontologies that comprise ROoST and the relationships between them are presented. Relations with the reused patterns are also presented.

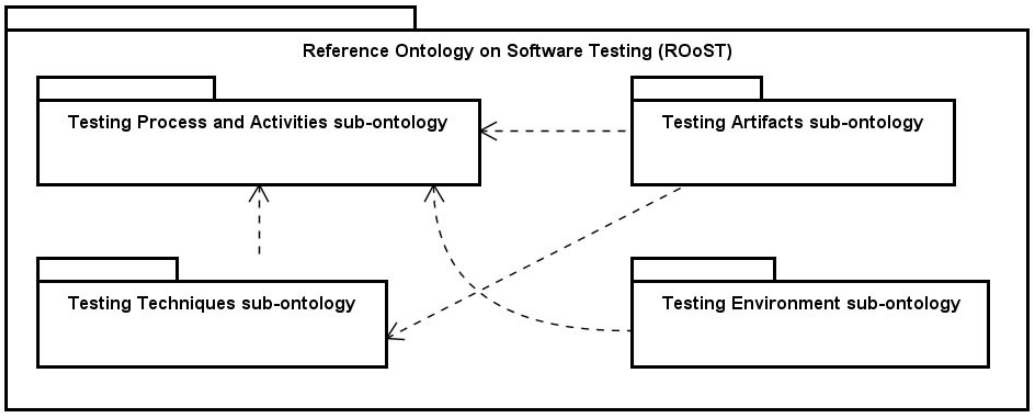

Figure 4.3 - ROoST: sub-ontologies

In the following subsections the ROoST sub-ontologies are presented as well as how

we applied the reused patterns in their development. Concepts reused from SP-OPL are shown in grey, and they are preceded by the pattern acronym (e.g., PAE::).

4.2.2 Testing Process and Activities sub-ontology

This sub-ontology addresses the competency questions *CQ1* to *CQ6*. To answer them, the PAE pattern was reused. PAE concepts were extended to the testing domain, as shown in Figure 4.4. *Testing Process Occurrence* is a subtype of *Specific Process Occurrence*, since a testing process occurs in the context of the entire software process (*General Process Occurrence*) of a *Project*. A testing process, in turn, is composed by testing activities, and thus *Testing Activity Occurrence* is considered a subtype of *Activity Occurrence*. As well as *Activity Occurrence*, *Testing Activity Occurrence* can be further divided into *Composite* and *Simple Testing Activity Occurrences*.

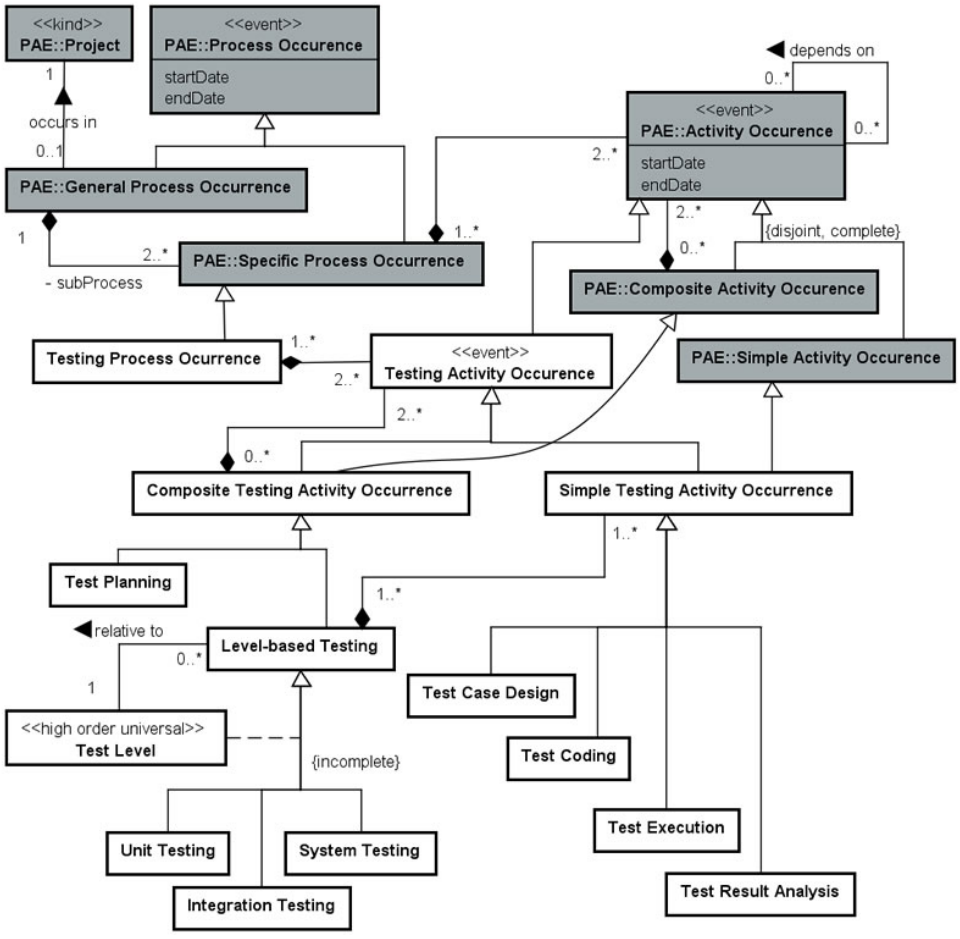

Figure 4.4 - ROoST´s Testing Process and Activities sub-ontology.

Besides specializing concepts, relationships were also specialized from PAE. For instance, in PAE, there is a whole-part relationship between *Specific Process Occurrence* and *Activity Occurrence*. The whole-part relationship between *Testing Process Occurrence* and *Testing Activity Occurrence* is a subtype of the former. Whenever a ROoST relationship is a subtype of another relationship defined in SP-OPL, the same name is used for both.

Looking at the literature (ABRAN et al., 2004; BLACK; MITCHELL, 2008; MATHUR, 2012), it is possible to say that the testing process consists of, at least, the following activities: *Test Planning*, *Test Case Design*, *Test Coding*, *Test Execution*, and *Test Result Analysis*. Thus, these activities were considered subtypes of *Testing Activity Occurrence*. Moreover, *Test Planning* was considered a *Composite Testing Activity Occurrence*. Although not shown in Figure 4.4, test planning involves several sub-activities, such as defining the testing process, allocating people and resources for performing its activities, analyzing risks, and so on. On the other hand, *Test Case Design*, *Test Coding*, *Test Execution* and *Test Result Analysis* were considered as *Simple Testing Activity Occurrences*. Test Planning activity not was decomposed into sub-activities in ROoST, since, by inspecting the literature, we could not reach a consensus regarding which sub-activities comprise Test Planning.

Software testing is usually carried out at different test levels (MATHUR, 2012). *Simple Testing Activity Occurrences* are grouped according to the *Test Level* to which they are related, forming *Level-based Testing* activity occurrence (*CQ6*). Thus, *Level-based Testing* is a subtype of *Composite Testing Activity Occurrence*. In Figure 4.4, the three most cited testing levels in the literature are made explicit: *Unit Testing*, *Integration Testing* and *System Testing*. However, there may be other, such as *Regression Testing*.

To answer *CQ1*, two axioms were defined in PAE that says that the relationship *occurs in* between *General Process Occurrence* and *Project* can be extended to the sub-processes and activity occurrences that compose the former.

$$\textbf{(A1)} \quad \forall gpo : GeneralProcessOccurrence; \quad p : Project, \quad spo : SpecificProcessOccurrence \quad occursIn(gpo, p) \wedge partOf(spo, gpo) \rightarrow occursIn(spo, p))$$

$$\textbf{(A2)} \quad \forall spo : SpecificProcessOccurrence; \quad p : Project, \quad ao : ActivityOccurrence \quad occursIn(spo, p) \wedge partOf(ao, spo) \rightarrow occursIn(ao, p))$$

4.2.3 Testing Artifacts sub-ontology

The Testing Artifacts sub-ontology addresses the competency questions *CQ7* to *CQ9*. To answer *CQ7* and *CQ8*, the WPPA pattern was reused. On the other hand, since in ROoST there is no interest in modeling the events representing the artifact participations, but only which artifacts were used and produced by a testing activity occurrence, only the derived relationships */uses* and */produces* were modeled, instead of modeling the artifact participations.

An important issue for ROoST is to describe the types of artifacts that are produced and used during the testing process. Thus, the Work Product Taxonomy (WPT) pattern was reused too. In WPT, an incomplete taxonomy of software artifacts is defined including, among others, the following kinds of artifacts: *Document*, which refers to artifacts consisting of textual statements usually associated with organizational patterns that define how they should be produced; *Code*, which concerns to portions of code written in a programming language; and *Data*, referring to data used or produced during the software process.

During the software testing process, several artifacts are used and produced. An important issue for ROoST is to precisely define the relationships between testing activities and artifacts (*CQ7* and *CQ8*), as well as the relationship between the artifacts (*CQ9*). In order to make this part of the testing domain conceptualization explicit, the relationships *uses* and *produces* from WPPA were specialized to link testing artifacts to the corresponding testing activities in which they were produced or used. Moreover, relationships between the testing artifacts were defined, as shown in Figure 4.5.

During *Test Planning*, a *Test Plan* is produced. In *Test Case Design*, different artifacts are used for deriving test cases, such as requirements specifications, use cases, diagrams, programs, and so on. *Artifacts* used for deriving test cases play the role of *Test Case Design Input*. The main outputs of a *Test Case Design* activity are *Test Cases*. A *Test Case* aims to test a *Code To Be Tested*, and specifies the *Test Case Input* and the *Expected Result*.

Test Case Input and *Expected Result* are roles played by *Data* in a test case, and are part of it. Whatever code fragments (such as programs, modules, and the whole system code) that have a *Test Case* designed for them play the role of *Code To Be Tested*. It is worth highlighting that "role" in this work is used in the context of UFO, i.e. a role is an anti-rigid specialization of a sortal such that the specialization

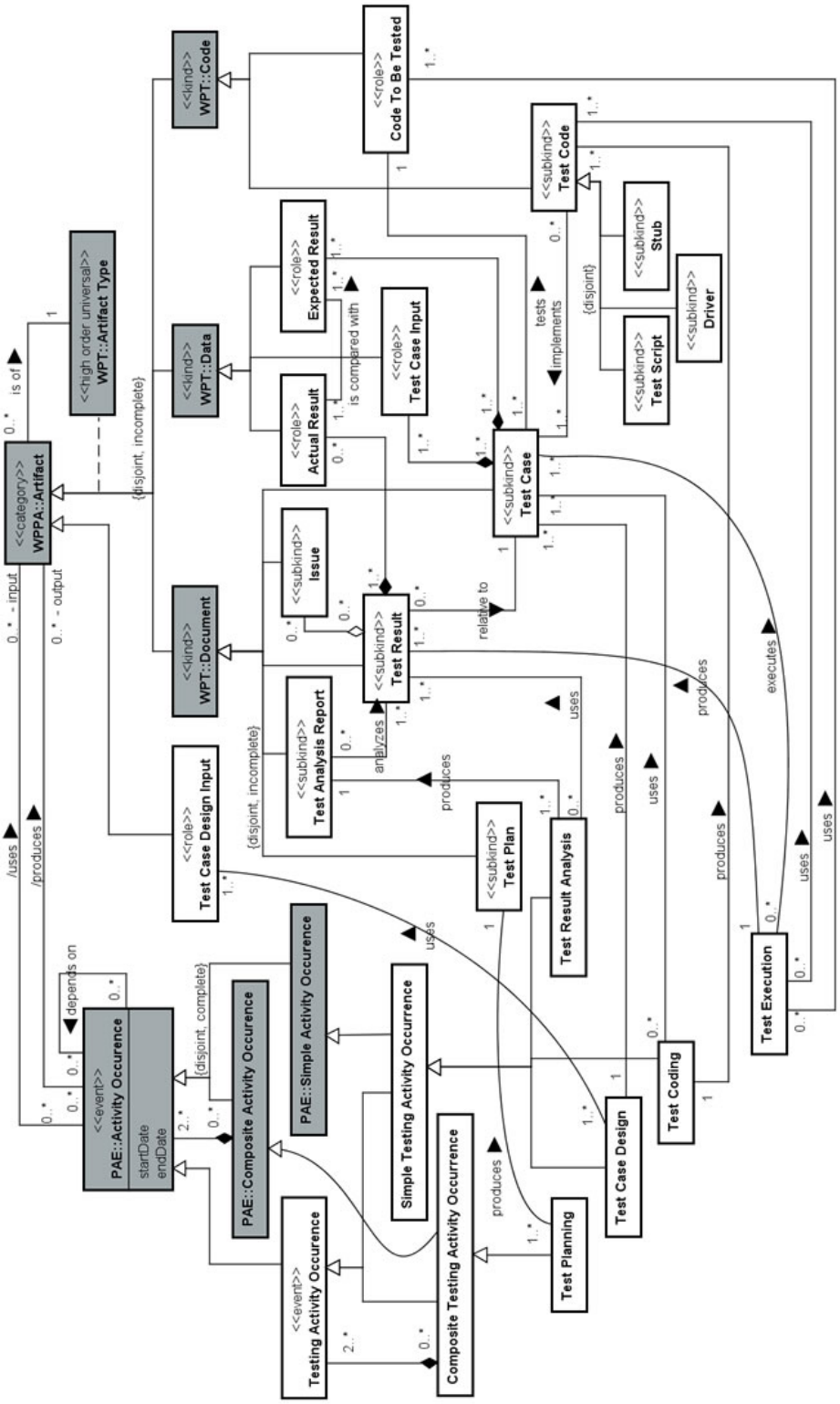

Figure 4.5 - The ROoST's Testing Artifacts sub-ontology.

condition is a relational one (GUIZZARDI, 2005). Take as an example the role *Code To Be Tested*. It is an anti-rigid specialization of *Code* (a kind in UFO), such that the specialization condition is to be the target code of a *Test Case* (*tests* relation). The relational property of being the code to be tested by a *Test Case* is part of the very definition of the role *Code To Be Tested*. Whenever a concept in ROoST is stereotyped with the <<role>>, this view applies.

During a *Test Coding* activity occurrence, *Test Cases* are used to derive *Test Code* that implements them. *Test Code* is a portion of code that is to be run for executing a given set of test cases. There are three main subtypes (*subkind*) of *Test Code*: *Test Script*, *Driver* and *Stub*. A *Test Script* corresponds to a sequence of actions for the execution of a *Test Case*. A software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results is called test *Driver*. And the *Stub* is a computer program statement substituting for the body of a software module that is or will be defined elsewhere.

Test Execution requires as input the *Test Code* to be run and the *Code To Be Tested*. If a *Test Execution* executes a *Test Case*, then the *Test Case* should use a *Test Code* that implements *Test Case*, and *Test Execution* should also use a *Code to be Test* that is tested by *Test Case* (see Axiom A3). As an output of this activity, *Test Results* are produced. A *Test Result* is relative to a *Test Case*. Following this relationship, it is possible to know the *Test Case Input* and *Expected Result* to which an *Actual Result* must be compared during *Test Result Analysis* (see axiom A4). *Actual Result* is the role played by *Data* when it is part of a *Test Result*.

$$\textbf{(A3)} \quad \forall te : TestExecution, \quad tc : TestCase \quad executes(te, tc) \rightarrow (\exists tco : TestCode, \quad ctbt : CodeToBeTested) \quad uses(te, tco) \wedge implements(tco, tc) \wedge uses(te, ctbt) \wedge tests(tc, ctbt)$$

$$\textbf{(A4)} \quad \forall ar : ActualResult, \quad er : ExpectedResult \quad comparedWith(ar, er) \rightarrow (\exists tc : TestCase, tr : TestResult) \quad partOfTestResult(ar, tr) \wedge relativeTo(tr, tc) \wedge partOfTestCase(er, tc)$$

A test execution can run and achieve a result (*Actual Result*), but it can also fail, generating an *Issue*. An *Issue* may report an incident. Incidents may be defects or bugs, but may also be perceived problems, anomalies that are not necessarily defects. In an incident, what is initially recorded is the information about the failure (not the defect) that was generated during test execution. The information about the defect that caused that failure would come to light when someone (e.g. a developer) begins

to look into the failure, but this is out of the scope of software testing. According to IEEE (1990), *Issue* is used to represent any event found during the execution of a software test that requires investigation. Thus, a *Test Result* contains either an *Actual Result*, or an *Issue*, or both. Moreover, a *Test Result* must include one of them, as defined by the following axiom:

$$\textbf{(A5)} \quad \forall tr : TestResult \rightarrow \exists \quad art : Artifact \quad (ActualResult(art)) \vee Issue(art)) \wedge partOf(art, tr)$$

Finally, during a *Test Result Analysis*, *Test Results* are analyzed and a *Test Analysis Report* is produced.

WPPA pattern also defines an important axiom for ROoST to answer *CQ5*. This axiom says that if an artifact *art* is an output of an activity occurrence *a1*, and *art* is also an input to another activity occurrence *a2*, then *a2* depends on *a1*.

$$\textbf{(A6)} \quad \forall a1, a2 : ActivityOccurrence, \quad art : Artifact \quad (produces(a1, art) \wedge uses(a2, art) \rightarrow dependsOn(a2, a1))$$

From this axiom, it is possible to infer important dependencies between testing activities, namely: *Test Coding* depends on *Test Case Design*; *Test Execution* depends on *Test Coding*; *Test Result Analysis* depends on *Test Execution*. Moreover, the *depends on* relationship is transitive (A7). Thus, can say, for instance, that *Test Execution* also depends on *Test Case Design*.

$$\textbf{(A7)} \quad \forall a1, a2, a3 : ActivityOccurrence \quad (dependsOn(a3, a2) \quad and \quad dependsOn(a2, a1) \rightarrow dependsOn(a3, a1))$$

4.2.4 Testing Techniques sub-ontology

This sub-ontology addresses the competency questions *CQ10* to *CQ12*. In order to answer them, the PRPA and Procedure Taxonomy (PRT) patterns were reused. According to the PRT pattern, *Procedures* are classified into: *Guideline*, *Method* and *Technique*. According to the PRPA pattern, *Procedures* can be adopted to support the accomplishment of *Activity Occurrences*. Analogously to WPPA, PRPA includes a concept for the events representing procedure participations in activity occurrences. However, since in ROoST there is no interest in representing those events, but only procedures were adopted by a testing activity occurrence, only the relationship */adopts* was worked, as shown in Figure 4.6.

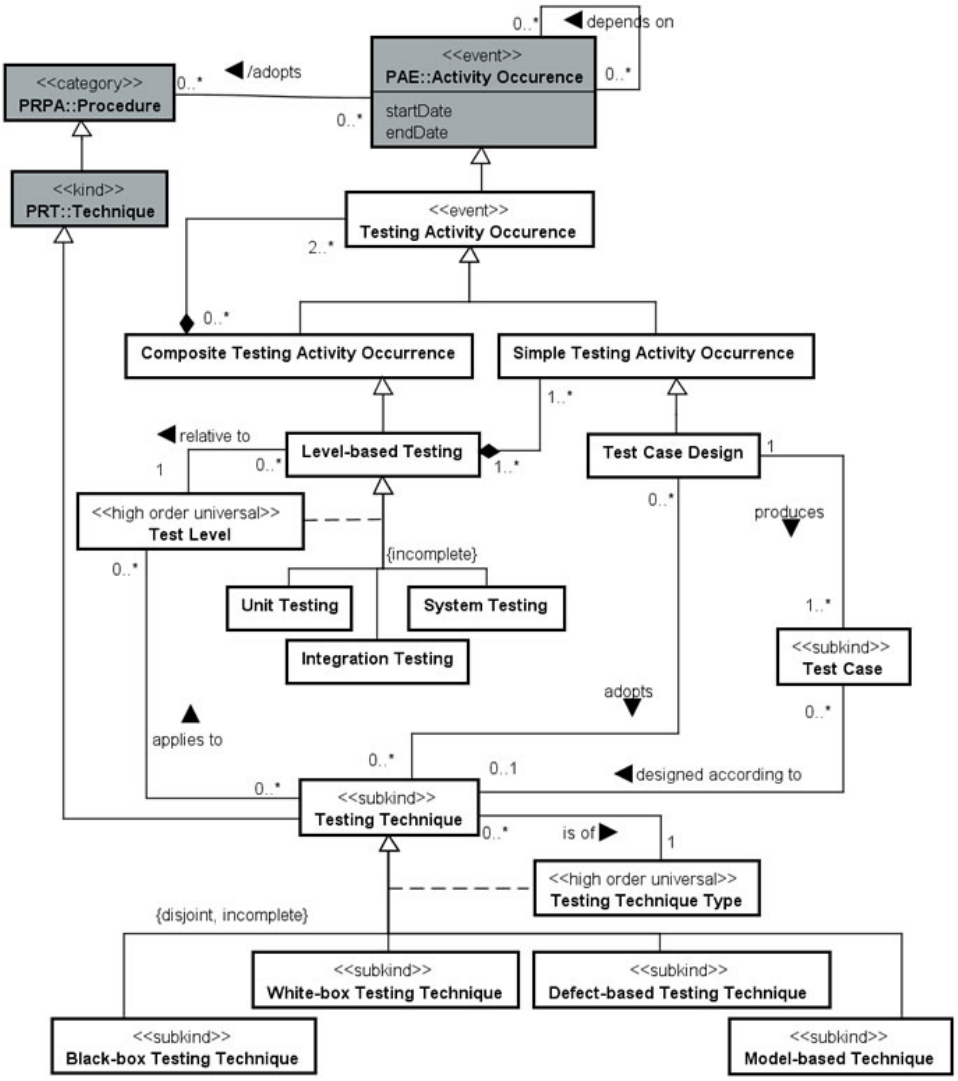

Figure 4.6 - ROoST's Testing Techniques sub-ontology.

To answer *CQ10*, only one kind of procedure (*Technique*) was needed, and thus this concept was specialized as *Testing Technique*. There are several subtypes of *Testing Technique*, among them: *Black-box*, *White-box*, *Defect-based*, and *Model-based Testing Techniques*. These testing techniques can be adopted by activity occurrences of the type *Test Case Design*.

Some testing techniques are more appropriate to certain test levels. To answer *CQ11*, a relationship between *Testing Technique* and *Test Level* was introduced. *Black-box Testing Techniques*, for example, apply to all test levels. *White-box Testing Techniques*, on the other hand, are suitable only for *Unit Testing* and *Integration Testing*. They are not suitable for *System Testing*, because it is difficult in practice to derive

test cases based on the source code when the entire system is considered (MATHUR, 2012). To represent this behavior in which a Level-Based Testing can only adopt Testing Techniques applicable to Level Test, the axiom A8 was defined. Furthermore, *Unit Testing*, *Integration Testing*, and *System Testing* are typical instances of *Test Level*, which is the criterion for the generalization set of *Level-based Testing*.

$$\textbf{(A8)} \quad \forall TestCaseDesign, \quad lbt : Level - baseTesting, tt : TestingTechnique, \quad tl : TestLevel \quad adopts(tcd, tt) \wedge partOfLevelBasedTesting(tcd, lbt) \wedge relativeTo(lbt, tl) \rightarrow appliesTo(tt, tl)$$

Finally, for designing a specific test case, a testing technique may be applied. Thus, for answering *CQ12*, a relationship between *Testing Technique* and *Test Case* was introduced, in order to link a test case to the testing technique applied in its design. If a testing technique was used in a certain test case design, then the test case design activity that produced this test case should have adopted this testing technique, as defined by the following axiom:

$$\textbf{(A9)} \quad \forall tc : TestCase, \quad tt : TestingTechnique, \quad tcd : TestCaseDesign \quad designedAccordingTo(tc, tt) \quad and \quad produces(tcd, tc) \rightarrow adopts(tcd, tt))$$

4.2.5 Testing Environment sub-ontology

The Testing Environment sub-ontology addresses the competency questions *CQ13* to *CQ19*. It was developed using the patterns RPA and HRPA, as shown in Figure 4.7. According to the RPA pattern, during an *Activity Occurrence*, *Resources* concept is used. In this pattern, two important types of resources are considered, since they are very relevant in the context of software processes: *Hardware Resource* refers to the use of a *Hardware Equipment* in an activity occurrence and *Software Resource* refers to the use of a *Software Product* in an activity occurrence.

Test Environment is defined for a *Project* and, in turn, is composed by *Test Hardware Resource*, *Test Software Resource* and *Human Resource*. *Test Hardware Resource*, *Test Software Resource* are the roles played by a *Resource* and are used by *Testing Activity Occurence*. *Testing Activity Occurence* also uses *Test Environment*. With respect to the test software resources, for answering *CQ17*, the concept *Software Product* was specialized as *Test Software Product*. There are several subtypes of *Test Software Product Type*, among them: *Test Management Tool, Test Execution Tool,* and *Incident Management Tools*.

Testing Activity Occurence is performed by *Human Resource*. In *Human Resource* different testing roles are contemplated (*CQ13*), such as *Test Leader, Test Case Designer* and *Tester*. Aspects related to human resource were built using the E-OPL patterns. The enterprise aspects addressed by E-OPL and used in Testing Environment sub-ontology are: Organization Arrangement, Team Definition, Institutional Roles and Human Resource Management. These aspects address problems related to how an organization is structured (FALBO et al., 2014). Patterns Multi-Organization Arrangement (MOAR), Organizational Teams (OTD), Team Roles (TEAR) and Team Allocation (TEAA) were reused. Concepts reused from E-OPL are shown in yellow, and they are preceded by the pattern acronym (e.g., MOAR::). The patterns used are presented in details in Annex A.

An *Organization* might have *Organizational Teams* (e.g., *Test Team*). An organizational team is composed by human resources that are its members at a given point in time. A *Team Allocation* associates the *Team Member* (the role a *Human Resource* plays when he/she is allocated to a Team) to the *Team*, in a given period of time (*CQ14*). When a *Human Resource* is allocated to a *Team* (becoming an Organizational *Team Member*), it also defines the *Human Role* (e.g. *Test Leader, Test Case Designer, Tester*) that he/she must play in that *Team Allocation*.

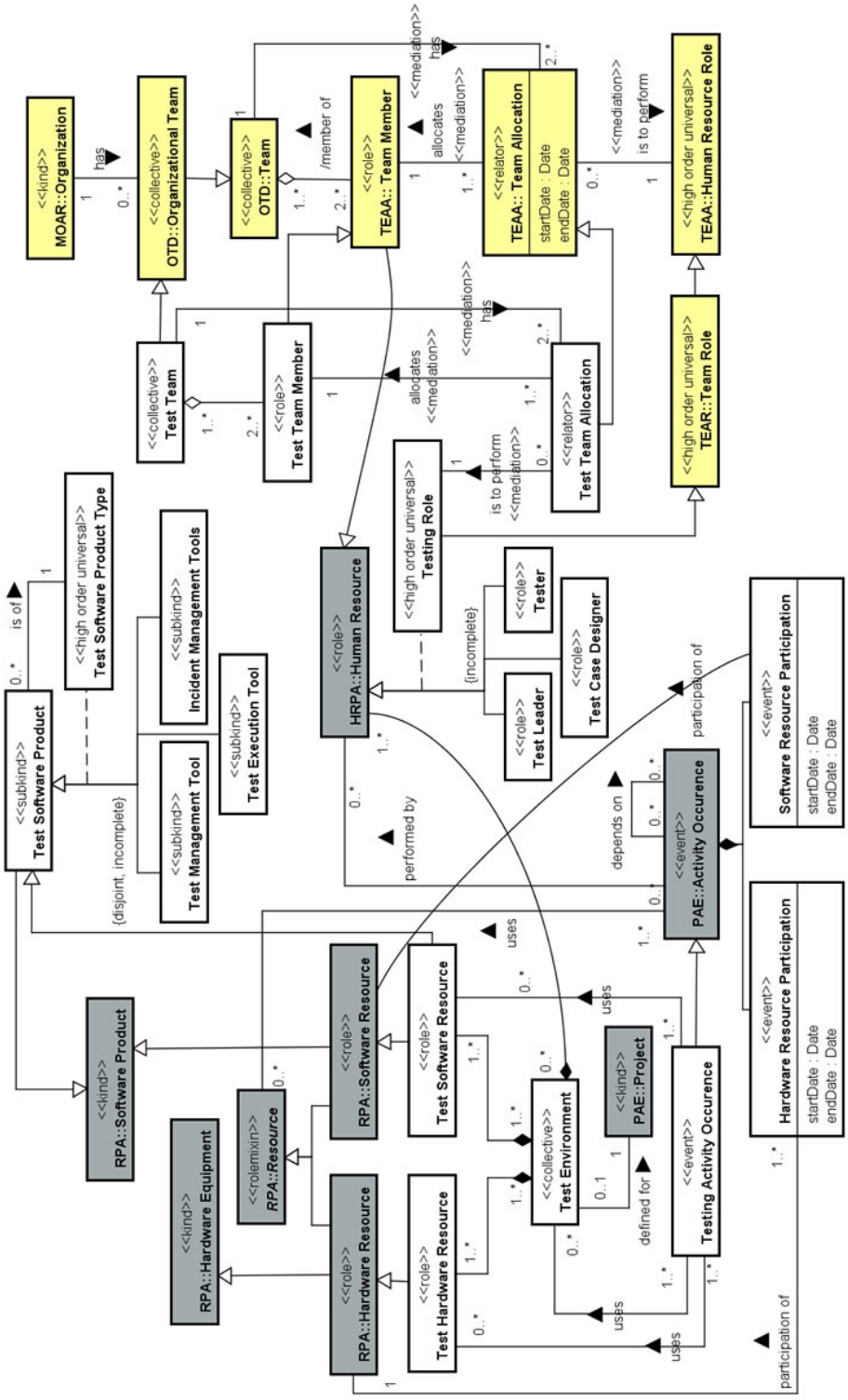

Figure 4.7 - ROoST's Testing Environment sub-ontology.

4.2.6 ROoST Evaluation

In order to evaluate ROoST, verification & validation activities for ontologies were performed. ROoST evaluation started with a verification activity, where it is checked if the concepts, relations and axioms defined in ROoST are able to answer the competency questions. Table 4.1 illustrates this verification process, showing which elements of the ontology (concepts, relations, properties and axioms) answer each Competency Questions (CQ).

Table 4.1 - ROoST Verification

CQ	**Concepts, *Relations* and Properties**	**Axioms**
CQ1	Testing Activity Occurrence *part of* Testing Process Occurrence Testing Activity Occurrence *subtype of* Activity Occurrence Testing Process Occurrence *subtype of* Specific Process Occurrence Activity Occurrence *part of* Specific Process Occurrence Specific Process Occurrence *part of* General Process Occurrence General Process Occurrence *occurs in* Project	A1, A2
CQ2	Test Planning *subtype of* Composite Testing Activity Occurrence Test Case Design *subtype of* Composite Testing Activity Occurrence Test Coding *subtype of* Composite Testing Activity Occurrence Test Execution *subtype of* Composite Testing Activity Occurrence Test Result Analysis *subtype of* Composite Testing Activity Occurrence	-
CQ3	Testing Process Ocurrence *subtype of* Process Occurence, which contains the properties *startDate* and *endDate*	-
CQ4	Testing Activity Occurrence *subtype of* Activity Occurrence, which contains the properties *startDate* and *endDate*	-
CQ5	Test Coding *depends on* Test Case Design Test Execution *depends on* Test Coding	A6, A7

Continues

Table 4.1 - Conclusion

CQ	**Concepts, *Relations* and Properties**	**Axioms**
	Test Result Analysis *depends on* Test Execution	
CQ6	Unit Testing *subtype of* Level-based Testing Integration Testing *subtype of* Level-based Testing System Testing *subtype of* Level-based Testing	-
CQ7	Test Planning *produces* Test Plan Test Case Design *produces* Test Case Test Coding *produces* Test Code Test Execution *produces* Test Result Test Analysis *produces* Test Analysis Report	A3, A4
CQ8	Test Case Design *uses* Test Case Design Input Test Coding *uses* Test Case Test Execution *uses* Test Code Test Execution *uses* Code To Be Tested Test Result Analysis *uses* Test Result	-
CQ9	Test Case Input and Expected are Result *part of* Test Case Test Code *implements* Test Case Test Result *is relative to* Test Case Actual Result and Issue are *part of* Test Result Test Analysis Report *analyzes* Test Result	A5
CQ10	Test Case Design adopts Testing Technique Black-box Testing Technique *subtype of* Testing Technique White-box Testing Technique Technique *subtype of* Testing Technique Defect-based Testing Technique *subtype of* Testing Technique Model-based Technique *subtype of* Testing Technique	-
CQ11	Testing Technique *applies to* Test Level	A8
CQ12	Test Case designed *according to* Testing Technique	A9
CQ13	Activity Occurrence *performed by* Human Resource	-
CQ14	Test Team Allocation *subtype of* Team Allocation, which contains the properties *startDate* and *endDate*	-
CQ15	Testing Activity Occurrence *uses* Test Hardware Resource	-
CQ16	Test Hardware Resource *subtype of* Hardware Resource and it *participation of* Hardware	-

Continues

Table 4.1 - Conclusion

CQ	Concepts, ***Relations*** and Properties	Axioms
	Resource Participation which contains the properties *startDate* and *endDate*	
CQ17	Testing Activity Occurrence *uses* Test Software Resource	-
CQ18	Test Software Resource *subtype of* Software Resource and it *participation of* Software Resource Participation which contains the properties *startDate* and *endDate*	-
CQ19	Test Environment *composed of* Test Hardware Resource Test Environment *composed of* Test Software Resource Test Environment *composed of* Human Resource	-

To validate ROoST, its concepts and relations were instantiated from an actual project, in order to check if the ontology was able to represent concrete situations of the real world. They were extracted from the Project Amazon Integration and Cooperation for Modernization of Hydrological Monitoring (ICAMMH) (BRAGA et al., 2009). Table 4.2 shows part of the instantiation.

Table 4.2 - ROoST Instantiation

Concept	Instance
Project	ICAMMH Project
Black-box Testing Technique	Equivalence partitioning, Boundary-value analysis (black-box techniques applied to derive test cases in the ICAMMH Project)
Test Case	Test Case *P01-256* [Collect by electronic media- Invalid date] (a test case produced in the ICAMMH Project)
Test Case Design Input	Use Case Specification "SAD_MCU_001-Customize Data Collection" (artifact that was used to derive the test case *P01-256*)
Test Case Input	2009-15-11 [Year-month-day/file .txt with month invalid for data collection in header] (input data to the test case *P01-256*)
Code To Be Tested	CollectFormUtil.java (Java class that is to be tested by the test case P01-256)

Continues

Table 4.2 - Conclusion

Concept	**Instance**
Test Code (Test Script)	*P01-256 Script* (a test script that implements the test case P01-256)
Actual Result	"Invalid file"

Besides creating the instances to the concepts and relationships as shown in Table 4.2, an implementation of ROoST in OWL was made available to become operational so that a proper evaluation may be conducted through consultations. It is important to get this kind of ontology structure due to the possibility of performing inferences. A transformation procedure for conversion requires the definition of a systematic mapping from ontoUML to a computational ontology language, such as OWL. For this a set of guidelines for mapping presented in (ZAMBORLINI, 2011) and (BARCELOS et al., 2013) were used. These guidelines allowed the mapping of static concepts represented in OntoUML for classes and properties directly encoded in OWL. The ontology in OWL can be manipulated by means of various tools. One of the most used is the Protégé[1] tool.

Figure 4.8 illustrates an example of how to transform ontoUML into OWL (BARCELOS et al., 2013).

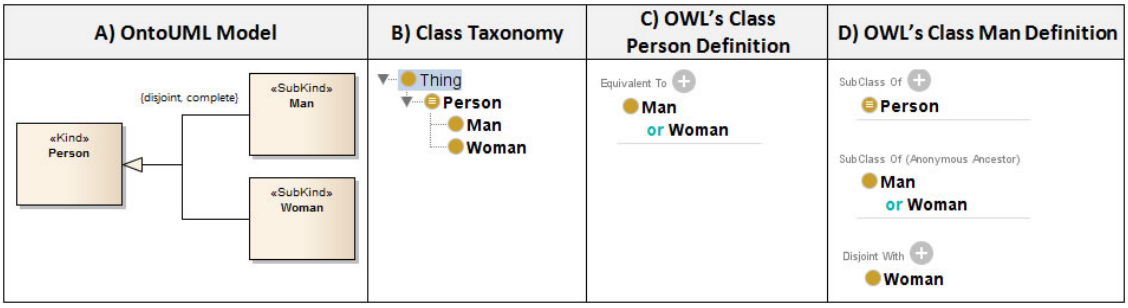

Figure 4.8 - Transformation of Generalization Sets
SOURCE: (BARCELOS et al., 2013)

Considering ROoST, presented below is a small fragment of the transcript of the ROoST conceptual model, modeled on language ontological level (ontoUML) for an epistemological level language (OWL). The concept *Artifact* exemplifies the classes

[1] http://protege.stanford.edu/

of type <<Category>>. This is the kind that groups rigid instances of classes with different principles of identity, in this case, instances of classes *Document, Data* and *Code* the <<kind>> type. *Artifact* is represented as a superclass, and as a generalization of class *Document, Data* and *Code*. The concepts of type <<kind>> are mapped as disjoint subclasses. The concepts of type <<subkind>> are also mapped as subclasses of their respective superclasses. For example, *Document* is <<kind>> and has <<subkind>> types with the following documents: *Test Plan, Test Result, Test Case* and *Issue*. These concepts are mapped to disjoint and equivalent in OWL subclasses, as shown in the fragment of Figure 4.9.

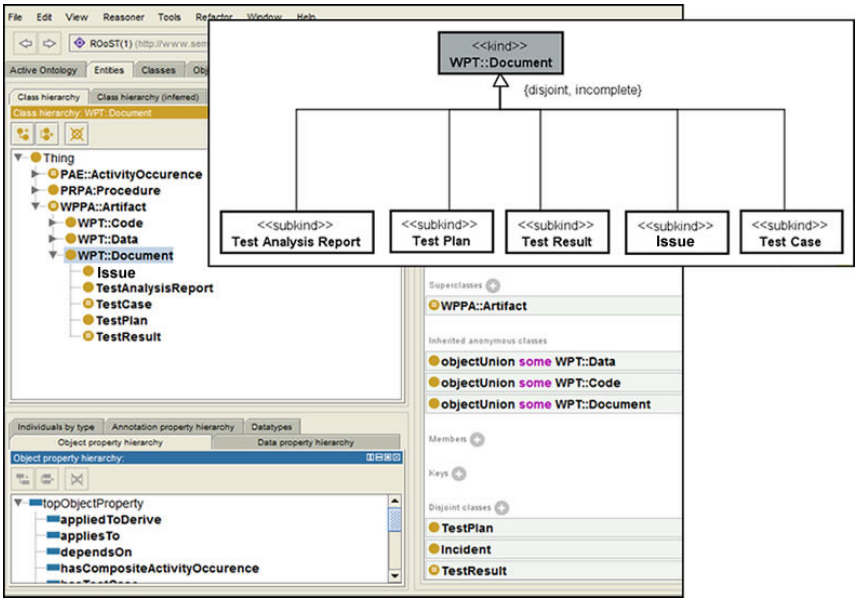

Figure 4.9 - Fragment on the mapping classes with stereotype <<category>>, <<kind>> and <<subkind>>

A second point is to represent the classes of type <<Role>> as OWL elements. In this case the classes *Actual Result, Test Case Input* and *Expected Result* are represented as a subclass of *Data* class, represented as a sortal type <<Kind>>. The fragment of Figure 4.10 presents this transcript.

Considering the relationship between classes (association between two or more classes), the fragment of Figure 4.11 presents some examples of the mapped relations. Among them:

- association between *Test Case* and *Test Result*
- aggregation of *Test Result* and *Issue*

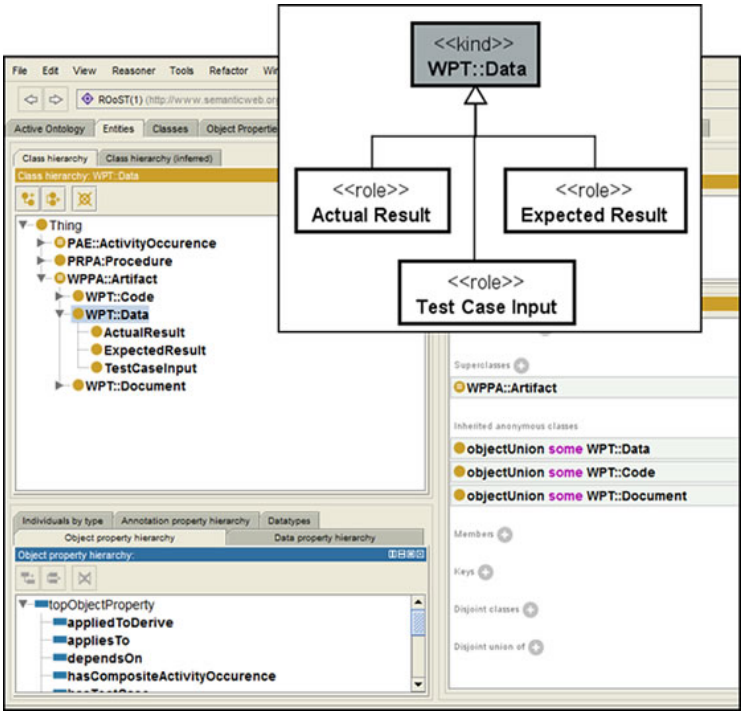

Figure 4.10 - Fragment on the mapping classes with stereotype <<Role>>

- composition between *Test Result* and *Actual Result*

Moreover, it is possible to check not only the relationship conversion but also the establishment of the properties and minimum cardinalities. According to Zamborlini (2011), it is necessary create mappings between classes and properties/attributes of the ontology. Some properties were included in the *TestCase* class in order to perform queries through query language Protocol and RDF Query Language (SPARQL). SPARQL can be used to query an RDF Schema or OWL model to filter out individuals with specific characteristics. Overall, SPARQL queries are composed of a series of triples where each format has triple <*Subject, Property, Object*>. For illustration purposes, presented below is an example of a query expression that returns all test cases, its own code and creation date. Figure 4.12 presents the query being performed by Protégé tool.

```
SELECT ?Tc ?Code ?Created
    WHERE { ?Tc :hasCode ?Code
          :hasDataCreated ?Created
    }
```

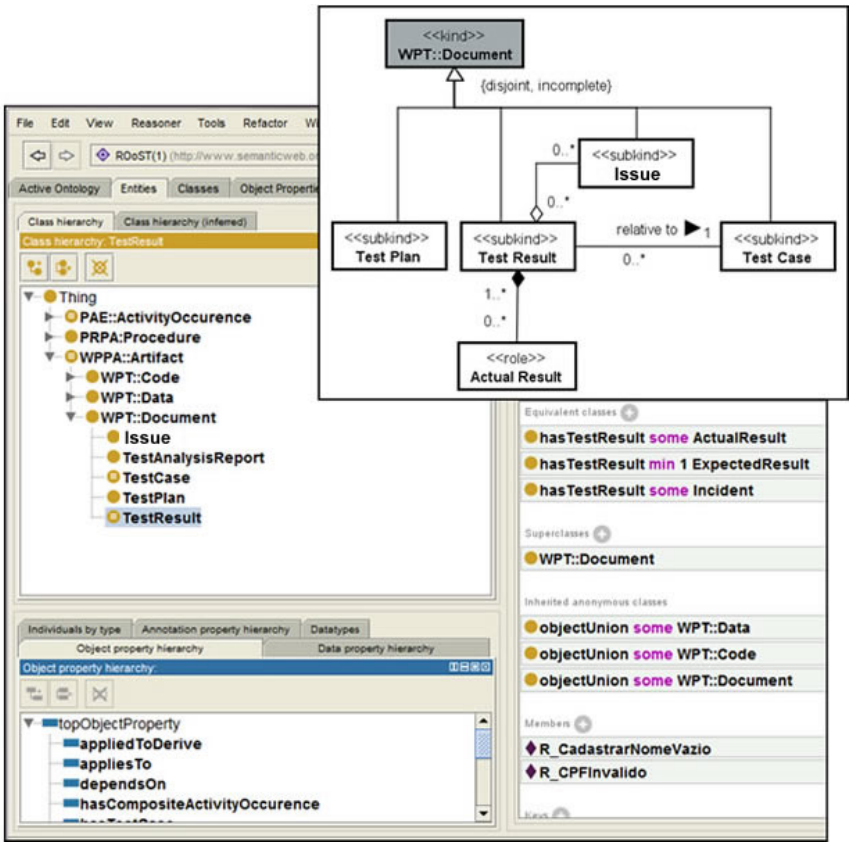

Figure 4.11 - Fragment on the mapping relations

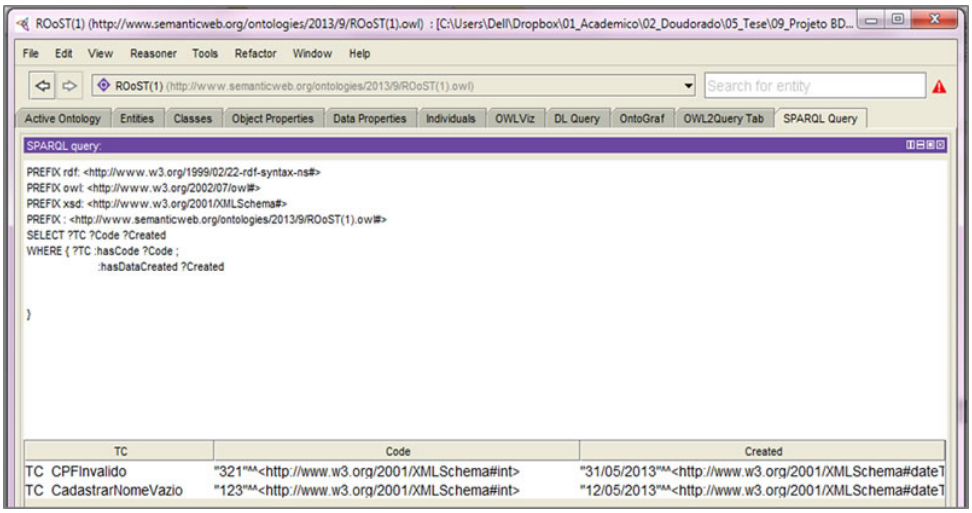

Figure 4.12 - SPARQL query on test cases

Finally, to evaluate ROoST, a comparison with the ontologies found in the SLR (Section 3.3) was performed considering the characteristics for "beautiful ontologies" proposed by D'Aquin and Gangemi (2011), namely: having a good domain coverage;

implementing an international standard; being formally rigorous; implementing also non-taxonomic relations; following an evaluation method; and reusing foundational ontologies. Table 3.6 presented in section 3.3.4 showed the comparison of the 12 ontologies found in 3.3. The following, in Table 4.3 is a summary of the ROoST is presented with the same characteristics.

Table 4.3 - Characteristics for "beautiful ontologies" in ROoST

Ontology	**Coverage**	**References**	**Engineering Method**
ROoST	In 4 subontologies: Testing Process and Activity, Artifact, Testing Technique, Test Environment (Hardware, software, team)	Several references, including international standards	SABiO

Evaluation Method	**Abstraction Level**	**Languages**	**Foundational Ontologies**
Verification & Validation activities	Conceptual models, implementation	OntoUML, OWL, axioms	UFO

The main distinguishing feature of ROoST when contrasted to other testing ontologies is that ROoST was developed taking characteristics of "beautiful ontologies" (D'AQUIN; GANGEMI, 2011) into account, as shown in Table 4.3. ROoST was developed following the SABiO method, which is a well-established method, used in several ontology development efforts (FALBO, 2004a). Moreover, ROoST was built by reusing and extending patterns of SP-OPL and E-OPL. Since SP-OPL and E-OPL are grounded in the UFO, ROoST inherits this foundational ground from these patterns. Further, concepts introduced in ROoST were also analyzed in the light of UFO. ROoST is a heavyweight modular ontology that was built considering several references, including international standards. It was evaluated from both verification and validation perspectives. Finally, concerning its coverage, ROoST covers aspects related to software testing process and its activities, artifacts that are used and produced by those activities, testing techniques for test case design, and, the software testing environment, including hardware, software and human resources.

4.3 Process for applying Knowledge Management in Software Testing

This section presents the T-KM Process, which provides directions for implementing KM initiatives in software testing supported by a KMS and which addresses the fourth goal of this thesis. The process, which is represented by the UML activity

diagram presented in Figure 4.13, is the second component of T-KM. As the figure shows, ROoST is used to support two steps of the process.

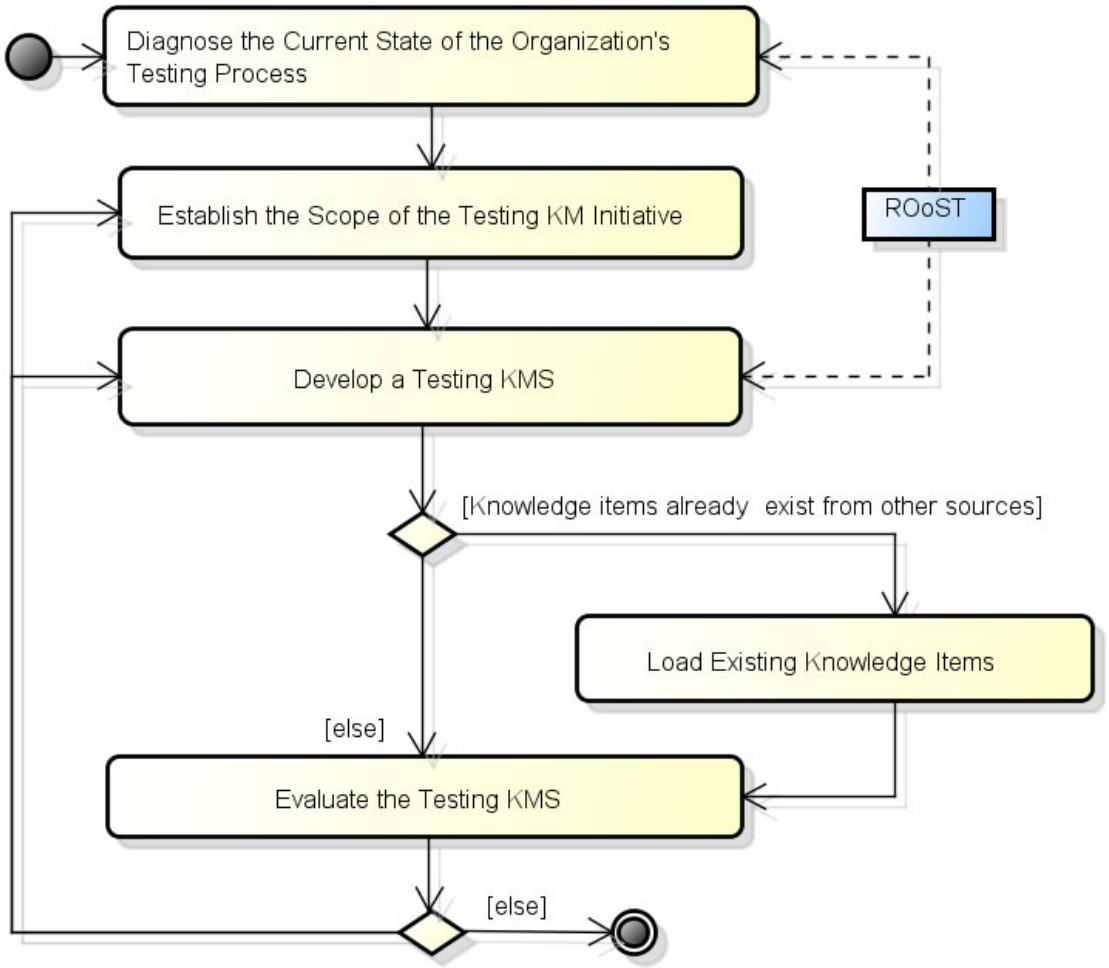

Figure 4.13 - T-KM Process

Following, each proposed step of process is presented in detail.

a) Diagnose the Current State of the Organization's Testing Process

The first step of the T-KM process is to make a diagnosis of the current state of the organization's testing process. This stage refers to investigating the existing knowledge within the testing process, in order to identify knowledge assets and understand how and where testing knowledge is developed and used in the organization. Once the knowledge items have been identified, organizations can then proceed to manage

them.

This step may be accomplished by performing surveys using questionnaires and/or interviews, to identify issues related to KM as an instrument for supporting the organization's testing process. It is suggested that this stage is accomplished raising the entire current state of software testing in the organization with respect to the following aspects: the testing process adopted, the activities of the testing process that are candidate to be the target of the KM initiative, the artifacts produced during this process, the testing techniques applied, the test levels contemplated by the process, the test environments adopted by the organization's software projects, among others. Aspects related to KM should also be investigated, such as the current KM practices applied in the testing process, organization's purpose of applying KM to software testing, problems related to testing knowledge in the organization, among others. Moreover, ROoST can be used in this step as the common vocabulary for supporting the analysis of the current situation, as well as to formulate the survey questions.

b) Establish the Scope of the Testing KM Initiative

Once the diagnosis of the current status of the testing process has been carried out, the next step is to establish the KM scope. This task requires knowing the organization needs. From the identified needs, the organization must define which activities of the testing process is to be supported, as well as which types of knowledge items are to be managed.

In this step it is suggested that organizations start with small KM initiatives. Thus, a major challenge for software organizations is to know which knowledge is really useful, and thus identify potential knowledge items among the several knowledge assets generated in the organization. The results of the survey are to be used in this step to help defining the scope of the testing KM initiative.

c) Develop a Testing KMS

This stage is concerned with the method for structuring the generated knowledge, that is, involves the KMS specification, which highlights the importance of a conducive and suitable organizational culture for facilitating knowledge sharing, creation and development in the organization. This step contains two main activities: the definition of the main requirements of a KMS which will manipulate the knowledge items identified and structuring of a knowledge repository.

As well as the activities of the testing process, the activities of the KM process that will be supported by the Testing KMS must be defined. We recommend to consider providing support to the following typical activities of a KM process, as discussed in Chapter 2 (Section 2.2): create knowledge items, assess knowledge items, search knowledge items, evaluate usefulness of available knowledge items, and maintain the knowledge repository.

With respect to requirements, they must be elicited and specified in this step. From the point of view of functional requirements, potential models to be created are use cases models, class diagrams, and state diagrams to model the behavior of knowledge items throughout its existence in the KMS. Other models can also be created according to the need. Non-functional requirements should also be addressed, such as security, usability, accessibility, etc.

ROoST is very useful in this step. ROoST can serve as the initial conceptual model of KMS, to be refined to include specific details of the testing KM initiative in hands. Specific information (attributes) should be identified, taking the characteristics of the organization's test environment into account, such as information provided by the tools used for managing software testing.

Furthermore, interoperability issues should also be analyzed in this step. Software tools that are part of the test environment can be integrated with the Testing KMS, in order to act together, interacting and exchanging data to obtain the expected results. In this context, possible knowledge items identified in these tools can be automatically converted/imported to the testing KMS.

During the design of the Testing KMS, developers should consider the platform in which the system is to be built, and non-functional requirements should be addressed. Once designed, the KMS should be coded and tested.

d) Load Existing Knowledge Items

For initially populating the knowledge repository of the Testing KMS, the organization should look for existing knowledge items. For instance, if the system has to manage test cases, existing test cases can be imported to the Testing KMS. The existing knowledge items should be mapped to conform with the structure of the knowledge repository. Finally, mechanisms for loading these items can be built to automate the loading process.

e) Evaluate the Testing KMS

Evaluation can be done by the organization to determine if the Testing KMS is feasible and meets its expectations. Improvements can be carried out continuously, implying a return to the previous steps. These improvements can be made continuously until the organization is satisfied with the Testing KMS achieved.

4.4 Final remarks about this chapter

In this chapter a T-KM framework was presented. T-KM is an ontology-based framework for guiding KM initiatives in the software testing domain, supported by KMS. This framework consists of a *Reference Ontology on Software Testing* (ROoST) for the software testing domain and a process with a set of directions for implementing a KMS in software testing organizations.

In the next chapter, as a proof of concept, a prototype of a Testing KMS has been developed under the T-KM framework. This is to show that applying the proposed framework major benefits such as to guide team members test the reuse of knowledge items, give support learning through the sharing of knowledge acquired when there is available knowledge, addition to supporting the management of skills and abilities of the organization members.

5 APPLICATION OF THE PROPOSED FRAMEWORK

In this chapter the Testing-Knowledge Management (T-KM) framework is employed to build a Knowledge Management System (KMS) for managing software testing knowledge, as a proof of concept of the feasibility of applying the proposed framework, which addresses the fifth goal of this thesis. Since there was no available testing organization to serve as a case study, T-KM was applied in a general scenario. Moreover, instead of building a KMS from scratch, the Knowledge Management (KM) Portal proposed in (COELHO, 2010) was extended to deal with software testing knowledge items. This portal (COELHO, 2010) was developed in the context of the Ontology-based software Development Environment (ODE) Project (FALBO et al., 2003) for supporting KM in Software Engineering.

As there was no available organization to run the entire process, in the diagnostic step, a general survey was conducted with questions that addressed aspects considered in the conceptualization of ROoST as well as aspects related to the mapping study presented in Section 2.3.1. Considering the survey results, and based on ROoST, the ODE´s KM Portal was extended, producing the Testing KM Portal. The survey and its use to define the scope of the testing KM initiative are discussed in Section 5.1. Data from an actual project were used as the basis for specifying the Testing KM Portal. Some details of its specification are presented in Section 5.3. The Testing KM Portal is presented in Section 5.4. The system was evaluated by project leaders from two actual scenarios. Section 5.5 discusses the system evaluation.

5.1 Diagnosis by means of a Survey

The mapping study presented in Section 2.3.1 showed that one of the major challenges in managing testing knowledge is to effectively integrate KM with software testing so that knowledge items can be shared and reused in testing organizations. Furthermore, managing testing knowledge is not an easy task, and thus it is better to start with a small-scale initiative. Firstly, it is necessary to identify essential knowledge items of a sub-topic of software testing to be dealt with the KMS. So, based on ROoST and the mapping study, in the diagnosis step, a survey was accomplished. The survey aim was to define a scenario to apply KM in software testing. Nine questions were defined, and the questionnaire was sent to testing experts. The questions are related to the mapping, as well as to the conceptualization described by ROoST, as Table 5.1 illustrates.

Table 5.1 - Survey Questions and their Relations with the Mapping Study and ROoST

Survey Questions	**Based on**
SQ1. In which activities of a Testing Process, is KM more useful? *SQ2.*In which activities of Testing Planning, is KM more useful?	**ROoST:** Testing Process and Activities sub-ontology
SQ3. A test environment consists of, among others, human resources, hardware and software. About which of these resources are more important to have available knowledge at the moment of defining the test environment?	**ROoST:** Testing Environment sub-ontology
SQ4. In which testing level is KM more useful?	**ROoST:** Testing Techniques sub-ontology
SQ5. What is the type of knowledge you consider to be more important during the software testing process?	**Mapping Study:** *RQ7.* (What are the types of knowledge items typically managed in the context of software testing?)
SQ6. Tacit knowledge can be made explicit, originating explicit knowledge. Regarding the types of knowledge items listed below, indicate the degree of importance of generating explicit knowledge from tacit knowledge.	**Mapping Study:** *RQ7*
SQ7. Regarding testing artifacts, which are the ones you judge to be more appropriate for reuse?	**Mapping Study:** *RQ7*, **ROoST:** Testing Artifacts sub-ontology
SQ8. What is the purpose of applying KM in Software Testing?	**Mapping Study:** *RQ6.* (What are the purposes of employing KM in software testing?)
SQ9. What benefits KM can bring to software testing?	**Mapping Study:** *RQ9.* (What are the main conclusions (benefits and problems) reported regarding applying KM in software testing?)

In total, 86 experts participated in the survey. Of these, 37 work directly with software testing (tester, test analyst, test designer, among others). 16 perform roles related to software development. 18 perform various underlying functions as a professor, researcher, consultant, among others. And 15 did not answer this question, because this question was considered optional.

Experts who answered the survey has an experience of at least 5 years. To establish the survey sample, groups interested in the area of software testing registered in the *LinkedIn*[1] network were invited to answer the survey.

The survey results were used to define the scope of the Testing KM Portal. Next the survey results to each question are briefly presented, as well as how these results were used to establish the scope of the testing KM initiative. Full survey and results are found in Appendix A.

SQ1. Importance of KM to Software Testing Process Activities

In this question, the following testing activities (defined in ROoST) were considered: (i) Planning Test; (ii) Design Test Case; (iii) Coding Test; (iv) Test Execution; and (v) Test Results Analysis. Figure 5.1 shows the percentage of answers per activity. "Test Case Design" (98.8%) and "Planning Test" (96.5%) have the largest representativeness.

SQ2. Usefulness of KM in sub-activities of Testing Planning

Figure 5.2 shows the percentage of answers per sub-activity of testing planning. The usefulness of KM for selecting the best testing techniques was recognized by 41% of the participants, since different types of test techniques determine different forms for selecting the test cases that will be used as input to the system under examination.

SQ3. Test Environment Resources

In the opinion of the experts, among the types of resource that comprise the test environment (human, software and hardware resources), "Human resource" is considered the most important at the time of setting the test environment, with 44% of the responses. Figure 5.3 shows the percentage of answers per test environment resources.

[1] http://www.linkedin.com/

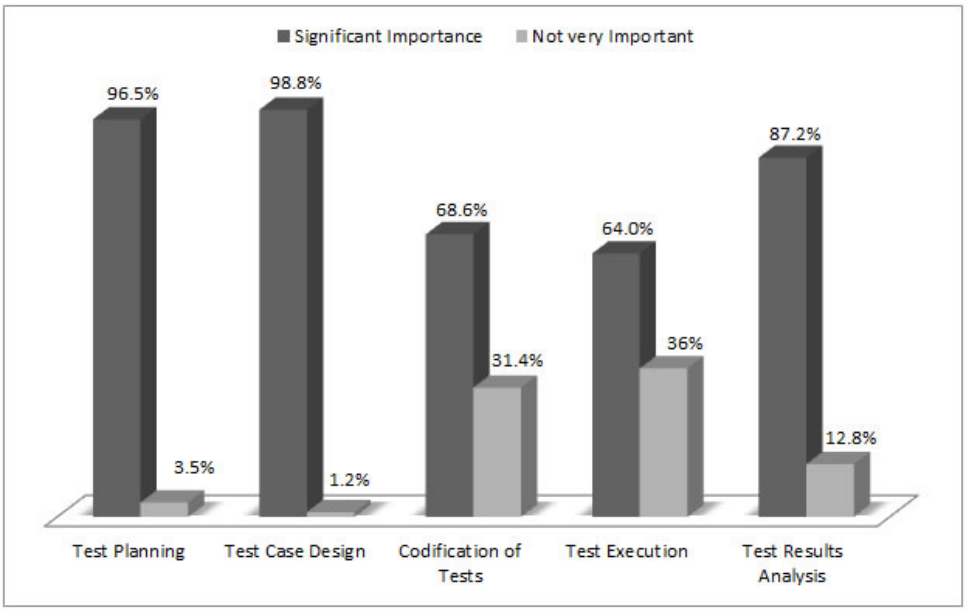

Figure 5.1 - Importance of KM to Software Testing Process Activities.

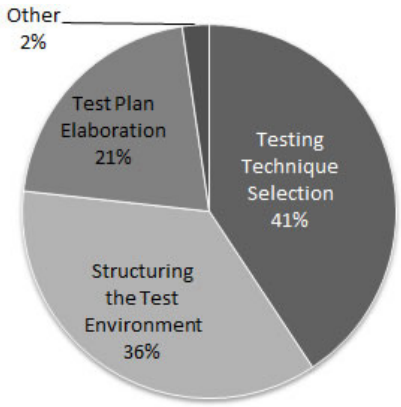

Figure 5.2 - Useful of KM in activities of Testing Planning

SQ4. Importance of KM to Test Levels

In this question, the three main levels of testing in the process of testing were considered: Unit Testing, Integration Testing and System Testing. The experts consider that in the software testing process, KM can be more useful in the System Testing level (49%). Figure 5.4 shows the percentage of answers per test level.

SQ5. Type of knowledge

In this question Tacit and Explicit Knowledge were considered. As a result of the survey, explicit knowledge was considered more important by most experts (69.8%).

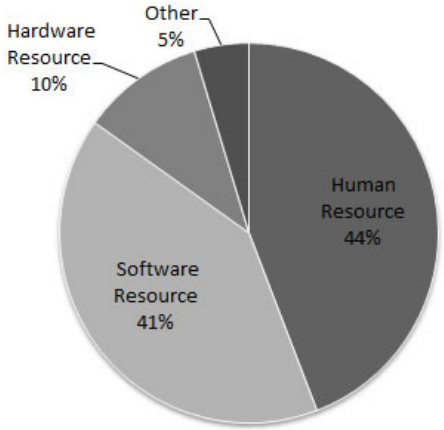

Figure 5.3 - Test environment resources

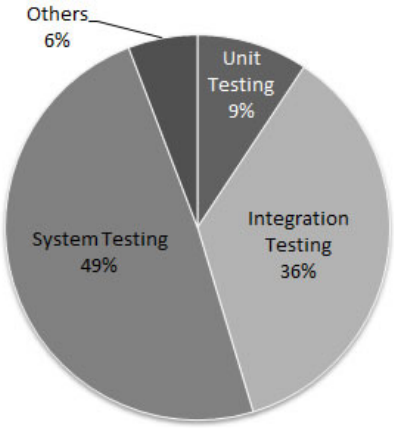

Figure 5.4 - Importance of KM to Test Level

Figure 5.5 shows the percentage of answers per type of knowledge.

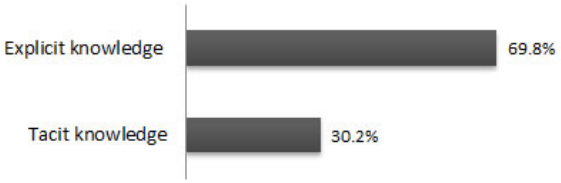

Figure 5.5 - Type of knowledge

SQ6. Making Tacit Knowledge Explicit

In the opinion of participants, "Individual Experiences" (95.3%) and "Communica-

tions between the members of the test team" (91.9%) are the types of tacit knowledge with more significant importance to generate explicit knowledge items. Figure 5.6 shows the percentage of answers.

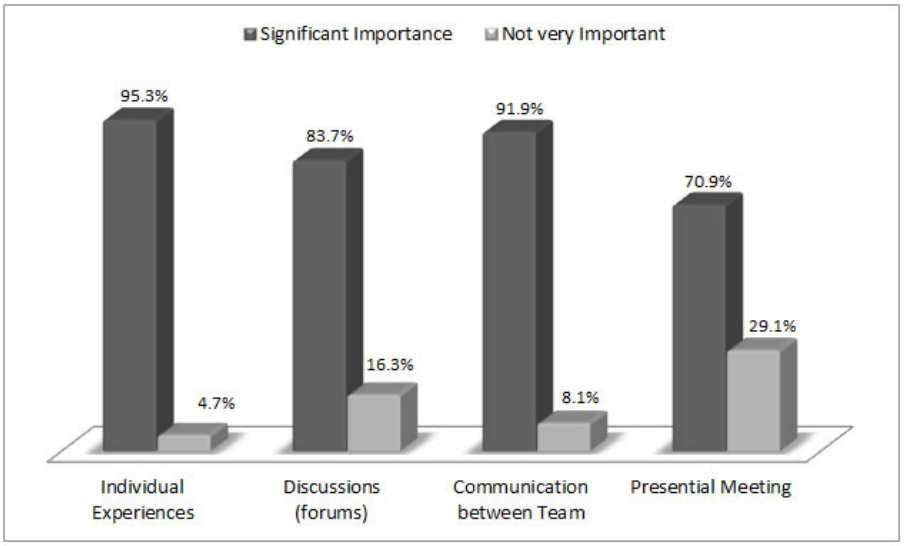

Figure 5.6 - Making Tacit Knowledge Explicit

SQ7. Artifacts more appropriate for reuse

Figure 5.7 shows the percentage of answers regarding the importance of reusing the test artifacts. In the opinion of the participants, "Test Plan" and "Test Case" artifacts are considered the most important artifacts for reuse in the software testing process, with percentages of 91.9% and 90.7%, respectively.

SQ8. Purpose of applying KM in Software Testing

Figure 5.8 shows the percentage of answers in relation to the purposes of applying KM in software testing. Considering the opinion of experts, "Improving the results quality" (28%) and "Reducing costs, time and effort" (26%) have the largest representativeness.

SQ9. Expected Benefits of applying KM in Software Testing

Finally, the benefits that KM can bring to software testing were analyzed. As Figure 5.9 shows, "increasing the process testing efficiency" (41%) and the "selecting and applying better techniques" (33%) were the most representative expected benefits.

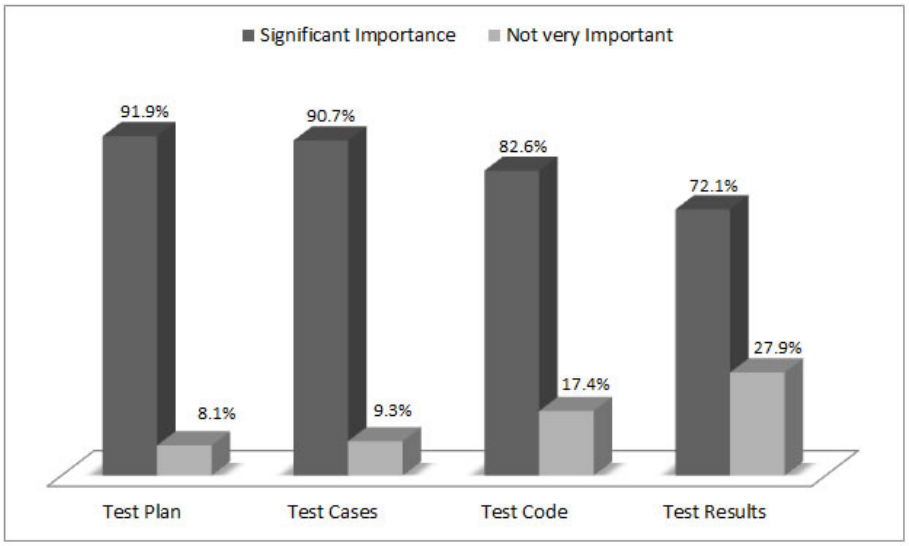

Figure 5.7 - Artifacts more appropriate for reuse

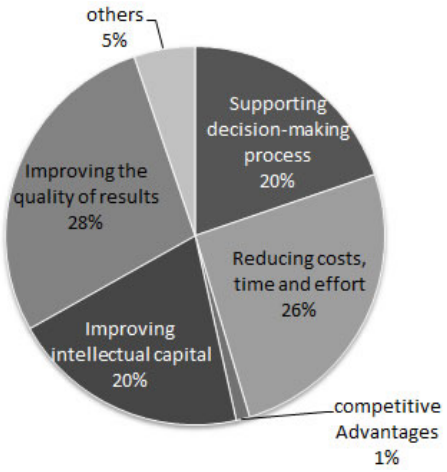

Figure 5.8 - Purpose of applying KM in Software Testing

By analyzing the survey results, some conclusions were found: (i) the experts identified test planning and test case design as being the activities in which KM would be most useful; (ii) Explicit knowledge was considered more important in software testing process. This proves what we raised in the systematic mapping: there is a difficulty in working with tacit knowledge. Moreover, it explains the increased interest in explicit knowledge; (iii) among the most targeted artifacts to reuse, test cases stood out with 90.7%; and (iv) the purposes for which experts are more interested in applying KM in software testing are related to improving the quality of results in software testing (28%), and reducing cost, time and effort spent in software testing (26%).

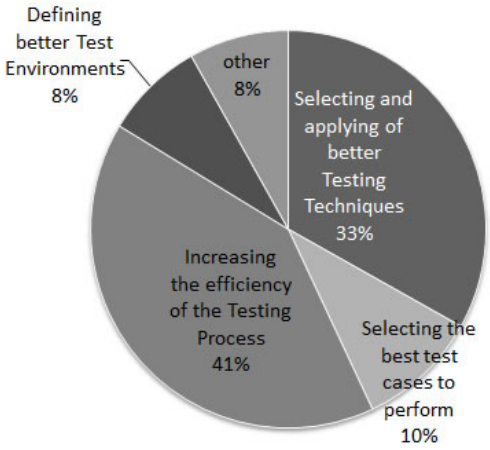

Figure 5.9 - Expected Benefits of applying KM in Software Testing

5.2 Definition of the Scope Testing KM Initiative

The results of the survey were then used to define the scope of the Testing KM Initiative. Considering the main findings of the survey, test case design was considered the software testing activity to be supported, and test cases the main knowledge item to be managed. All relevant information for designing test cases have to be considered in the scope of Testing KM Portal development. Concepts related to Test Case in ROoST were considered in the scope of initiative, namely: *Test Case Input, Expected Result, Test Result, Test Code* and *Testing Technique*. In addition, meetings with the project leaders from two actual projects also helped to understand the scope to be worked, as well as the experience of the author of this thesis as a member in one of the projects.

Moreover, the knowledge items listed below also were included. The first two (Lessons Learned and Knowledge regarding discussions) were already available as general items in ODE´s KM Portal. The last one was introduced from a previously conducted study related to knowledge discovery in data repositories for software testing (SOUZA; SANTOS, 2010).

- **Lessons Learned:** A Lesson Learned (LL) can be understood as knowledge acquired through experience in a particular situation. LL can be classified as best practices, errors/critiques and success factors. LLs are informal knowledge items that can be understood as ideas, facts, questions, point of view, decisions, among others. In addition, LL can also be classified as informative lessons, success or failure. Informative LLs explain how

to proceed in a given situation; lessons of success provide examples of problems that were solved in a positive way; and the failure lessons provide examples of negative responses to attempt to solve a problem and potential ways to cope up with the situation (O'LEARY, 1998a; NATALI, 2003).

- **Knowledge regarding discussions:** Discussion of ideas among organization members or questions answered by other sources of information may be submitted as knowledge items. Tools to support discussion among organization members, such as discussion forums, have been fundamental in KM environments (FISCHER; OSTWALD, 2001). Discussion forums become important tools for knowledge management for the following reasons: (i) Very useful knowledge can be generated and captured during argumentations (FALBO et al., 2004d), and (ii) a major challenge of KM is to convert tacit knowledge into explicit knowledge (NONAKA; KROGH, 2009; DAVENPORT; PRUSAK, 2000).

- **Mined items:** Useful knowledge can be identified, especially in large projects, from their data repositories. Knowledge Discovery in Databases (KDD) refers to the overall process of discovering useful knowledge from data. KDD is characterized as an interactive and iterative process, consisting of several interconnected steps (FAYYAD et al., 1996a). These steps start in a field definition, selection, preparation and data processing, until the step of data mining, where patterns can be found and analyzed to become useful knowledge.

5.3 Developing the Testing KM Portal

The conceptual model of ROoST was used as the starting point for specifying the Testing KM Portal. However, this conceptual model does not provide details regarding the properties of the concepts considered. Such properties can be identified according to the characteristics of the organization´s test environment, such as information provided by the tools used for managing software testing. Thus, information from one of the projects studied in this work (namely the Amazon Integration and Cooperation for Modernization of Hydrological Monitoring (ICAMMH) Project) were used as the basis for identifying attributes and specifying the Testing KM Portal.

Figure 5.10 shows use case diagram describing the functionality of the Testing KM Portal. The use cases detached in yellow were already available in ODE´s KM Portal

(COELHO, 2010). Use cases in white are testing-specific features included in this work. *Developer* is the main actor, representing all types of professionals involved in the software development process. *Knowledge Manager* represents a user with specific permissions, guaranteeing access to features inherent only to a Knowledge Manager.

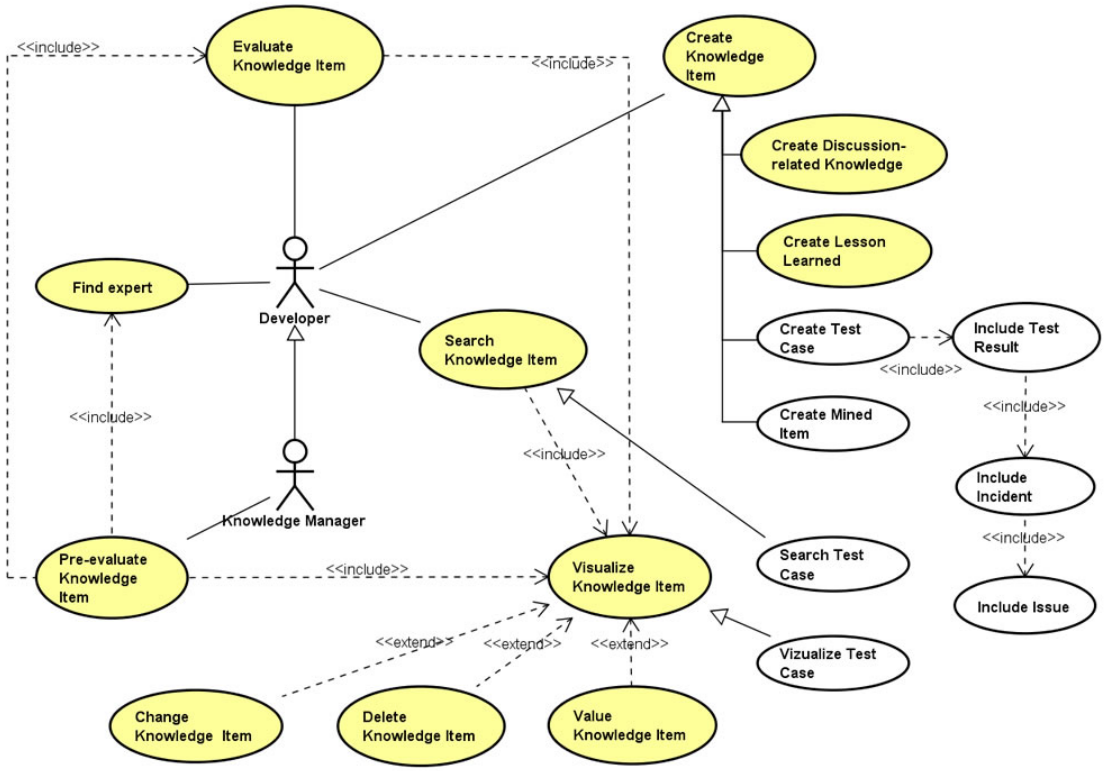

Figure 5.10 - Testing KM Portal Use Case Diagram

Next, the use cases are briefly described:

- **Create Knowledge Item:** This use case allows developers to create a knowledge item.
- **Create Discussion-related Knowledge:** This use case allows developers to register a Discussion-related Knowledge.
- **Create Lesson Learned:** This use case allows developers to register a Lesson Learned.
- **Create Mined Item:** This use case allows the developer to register a Mined Item.

- **Create Test Case:** This use case allows developers to register a Test Case.
- **Include Test Result:** This use case allows the developer to include a test result relative to a test case.
- **Include Issue:** This use case allows the developer to register an issue reporting an incident.
- **Include Incident:** This use case allows the developer to report an incident related to a test result.
- **Change Knowledge Item:** This use case allows the knowledge manager to change a knowledge item.
- **Delete Knowledge Item:** This use case allows the knowledge manager to delete a knowledge item.
- **Pre-evaluate Knowledge Item:** This use case allows the knowledge manager to pre-evaluate a knowledge item, making it available, rejecting it or selecting experts to evaluate it.
- **Evaluate Knowledge Item:** This use case allows a developer to make a detailed evaluation of a knowledge item, to support the knowledge manager in making decisions about whether the item should be approved or rejected.
- **Visualize Knowledge Item:** This use case allows developers to visualize the details of a knowledge item.
- **Visualize Test Case:** This use case allows developers to visualize the details of a test case.
- **Search Knowledge Item:** This use case allows the developer to search for knowledge items available according to informed parameters.
- **Search Test Case:** This use case allows the developer to search for test cases according to informed parameters.
- **Value Knowledge Item:** This use case allows the developer to value the utility of a knowledge item consulted.
- **Find Experts:** This use case allows the developer to find and select experts with a desired profile, as well as viewing the profiles of experts found. It works as a Yellow Pages system.

Figure 5.11 presents a package diagram showing the main packages of Testing KM Portal. The first one, the KM Portal package, is in fact, the package containing the classes from ODE´s KM Portal. The corresponding class diagram is presented in Figure 5.12 (COELHO, 2010). The second package, Testing KM Portal, contains the testing-specific classes developed in this work. The corresponding class diagram is presented in Figure 5.13. In Figure 5.13, classes in yellow are imported from the KM Portal package; classes in white are specifically developed to address the requirements of Testing KM Portal; enumerated types are shown in green.

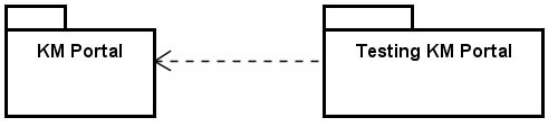

Figure 5.11 - Package Diagram of Testing KMS.

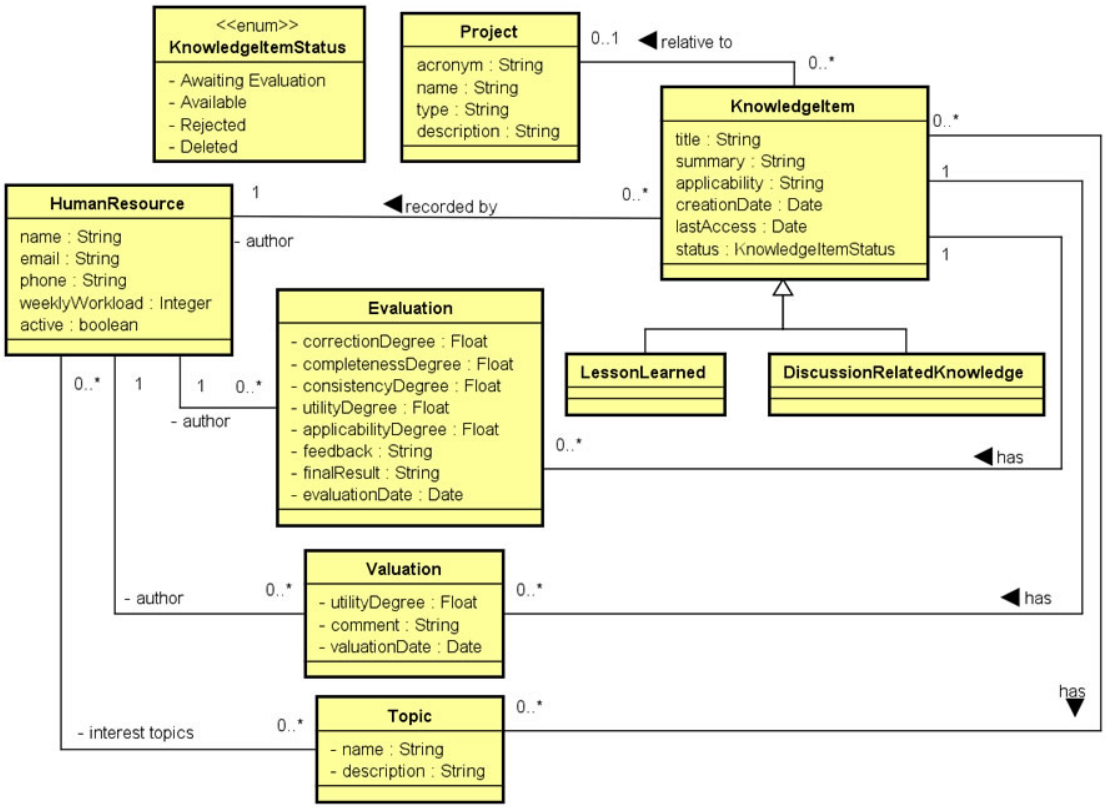

Figure 5.12 - Class diagram of KM Portal.
SOURCE: Adapted from Coelho (2010)

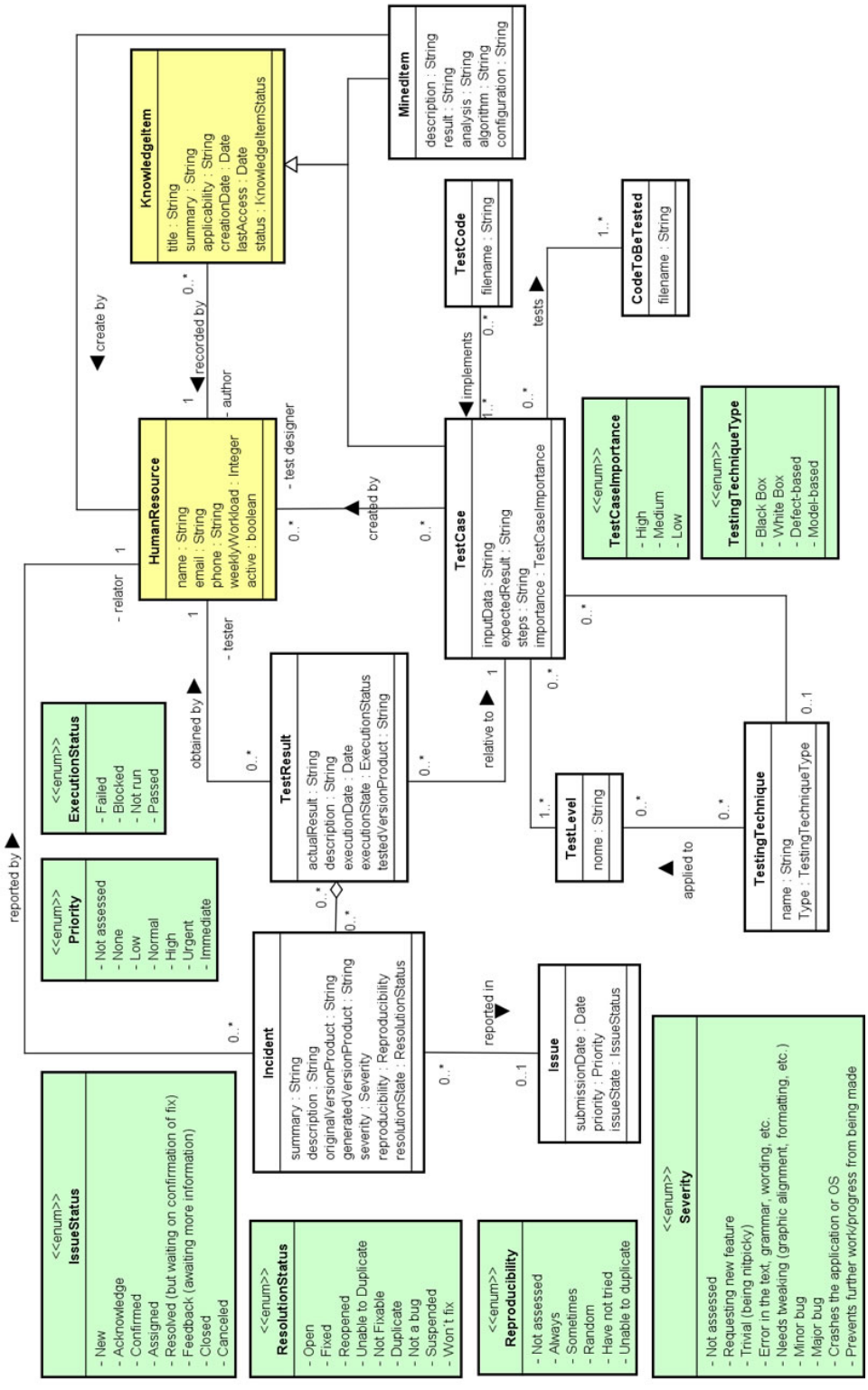

Figure 5.13 - Class diagram of Testing KM Portal.

After creating a knowledge item, an evaluation by experts should be performed (class *Evaluation*). Once a knowledge item becomes available to the organization, it can

be used and valued by any organization member (class *Valuation*). In a knowledge item creation, evaluation and valuation, it is necessary to inform who is the author (class *HumanResource*).

Knowledge items (class *KnowledgeItem*) can be of the following types: Lessons Learned (class *LessonsLearned*) and Discussion Related Knowledge (class *DiscussionRelatedKnowledge*). When creating a knowledge item, the developer can inform the project (class Project), a topic (class *Topic*) to which it is related, as well as specific information about each knowledge item.

Testing KM Portal extends KM Portal by allowing registering two new knowledge items: Test Cases (class *TestCase*), and Mined Items (class *MinedItem*). Next, these two types of knowledge items introduced by this work in the KM Portal are discussed.

5.3.1 Test Case

A test case contains information regarding its test levels (class *TestLevel*), the testing technique used to derive it (class *TestingTechnique*), the code it tests (class *CodeToBeTested*), the test code that implements this test case (class *TestCode*), the test results obtained when the test case is executed (class *TestResult*), and whether a test case result involves some incident (class *Incident*). As previously mentioned one issue (class *Issue*) may report an incident. For the test case result, it is also necessary to inform who exercised the test case, as well as who reported an incident.

The attributes of the classes in Figure 5.13, as well as the enumerated types shown in this figure were defined considering the ICAMMH Project. The ICAMMH Project was developed in a collaborative work involving the Brazilian Aeronautics Institute of Technology (*Instituto Tecnológico de Aeronáutica* - ITA) and the Brazilian National Water Agency (*Agência Nacional de Águas* - ANA), supported by the Brazilian Financial Foundation for Projects (*Financiadora de Estudos e Projetos* - FINEP). The project goal was to develop, for a water reference region, a pilot system for modernization and integration of telemetry points collected from hydrological data, as a basis for the management of water resources in the Amazon region.

The test environment for the ICAMMH Project comprised the following CASE tools: TestLink (TESTLINK, 2013) and MantisBT (MANTIS, 2013). Both are web tools, free for use and modification. TestLink is a web-based test management system. It offers

support for test cases, test suites, test plans, test projects and user management and reports. MantisBT is a bug (or defect) tracking system. However, it is often configured by users to serve as a more generic issue tracking system and project management tool. In the case of the ICAMMH Project, MantisBT was customized to deal with two categories of requests: activity-related requests and defect-related requests.

In the context of ICAMMH Project, an integration scheme between TestLink and MantisBT was used. TestLink has the capability to integrate with MantisBT, allowing for a test case to be associated to a defect-related request. Thus, all incidents that were registered in MantisBT, as a defect-related request, were conditioned to the existence of a test case in TestLink.

The ICAMMH Project has already been finalized. Nevertheless, the information managed by the tools that supported its testing process helped to identify important information (attributes) in addition to what is defined by ROoST. An analysis of the attributes from each tool was undertaken in order to identify relevant attributes for KM. Tables 5.2 and 5.3 show the main attributes considered from TestLink and MantisBT, respectively.

Table 5.2 - Attributes TestLink

Attribute	**Values Defined**	**Description**
Execution Status	Failed, Blocked, Not run, Passed	Defines the execution status of each test case.
Test Case Importance	Low, Medium, High	Defines the importance level for each test case created.

Table 5.3 - Attributes MantisBT

Attribute	**Values Defined**	**Description**
Priority	None, Low, Normal, High, Urgent, Immediate, Not assessed	Defines the priority level to correct an incident.

Continues

Table 5.3 - Conclusion

Attribute	Values Defined	Description
Severity	Requesting new feature, Trivial (being nitpicky), Error in the text, grammar, wording, etc., Needs tweaking (graphic alignment, formatting, etc.), Minor bug, Major bug, Crashes the application or OS, Prevents further work/progress from being made, Not assessed	Defines the severity level of an incident found.
Reproducibility	Always, Sometimes, Random, Have not tried, Unable to duplicate, Not assessed	Defines if the incident may be reproduced.
Incident Status	New, Acknowledge, Confirmed, Assigned, Resolved (but waiting on confirmation of fix), Feedback (awaiting more information), Closed, Canceled	Refers to the status of handling an incident.
Resolution Status	Open, Fixed, Reopened, Unable to Duplicate, Not Fixable, Duplicate, Not a bug, Suspended, Won´t fix	Defines the resolution status for a given request (activity or incident). This thesis takes into account only the records related to defects.

5.3.2 Mined Item

Data mining was performed on ICAMMH data. A preliminary analysis of this data mining was performed in Souza and Santos (2010) considering the databases of TestLink and Mantis tools. However, in this thesis, the same data in a different structure stored in the repository of knowledge of Testing KM Portal were used. Attributes related to knowledge items of type Test Case were considered.

For creating the mined items, the method of Rule Association was used along with the Apriori algorithm (WITTEN et al., 2005). The Association Rule method aims

to identify patterns of behavior to the set of data that often occur jointly in the database and form rules from these sets. The association rules, when applied to a data set, allow to find rules of the type of $X \rightarrow Y$, that is, transactions of the database which contain X tend to contain Y (WITTEN et al., 2005). One of the algorithms to the better known association rules is the Apriori algorithm. It can work with a large number of attributes, generating various combinations among them. For the generation of association with the Apriori algorithm, the tool Waikato Environment for Knowledge Analysis (WEKA) was used (WITTEN et al., 2005). WEKA is a collection of machine learning algorithms for data mining tasks. A brief explanation of how this item can be generated is given below.

Considering only those test cases that failed, 415 records were returned from a query in the knowledge repository. Table 5.4 presents the first 20 returned and 9 attributes considered in this data mining, corresponding to classes: Human Resource, Test Case, Incident, Issue.

Table 5.4 - Attributes analyzed (first 20 records)

Test Case Author	**Execution Author**	**Importance**	**Severity**	**Reproducibility**	**Issue Status**	**Priority**	**Resolution Status**
7	7	High	Minor Bug	Always	Closed	Normal	Fixed
7	7	High	Major Bug	Always	Closed	Normal	Fixed
7	7	High	Minor Bug	Always	Closed	Normal	Fixed
7	7	High	Crashes the application or OS	Always	Closed	High	Fixed
7	7	High	Major Bug	Always	Closed	Normal	Fixed
7	7	High	Major Bug	Always	Closed	High	Fixed
7	7	High	Major Bug	Always	Closed	High	Fixed
7	7	High	Major Bug	Always	Closed	High	Fixed
7	7	High	Major Bug	Always	Closed	High	Fixed
7	7	High	Minor Bug	Always	Resolved	Normal	Fixed
7	7	High	Major Bug	Always	Closed	Normal	Fixed
7	7	High	Major Bug	Always	Resolved	High	Fixed
7	2	High	Minor Bug	Have not tried	Resolved	Normal	Fixed
7	7	High	Mijor Bug	Always	Closed	Normal	Fixed

Continues

Table 5.4 - Conclusion

Test Case Author	Execution Author	Importance	Severity	Reproducibility	Issue Status	Priority	Resolution Status
7	2	High	Minor Bug	Have not tried	Closed	Normal	Fixed
7	2	High	Minor Bug	Have not tried	Closed	Normal	Fixed
7	2	High	Minor Bug	Have not tried	Resolved	Normal	Fixed
7	7	High	Minor Bug	Always	Closed	Normal	Not a bug
7	2	High	Major Bug	Have not tried	Closed	Normal	Fixed
7	2	High	Minor Bug	Have not tried	Resolved	Normal	Fixed

Figure 5.14 shows a WEKA screen with the set of data already preprocessed and imported. It is possible to see the attributes, number of instances and some statistical information of each class.

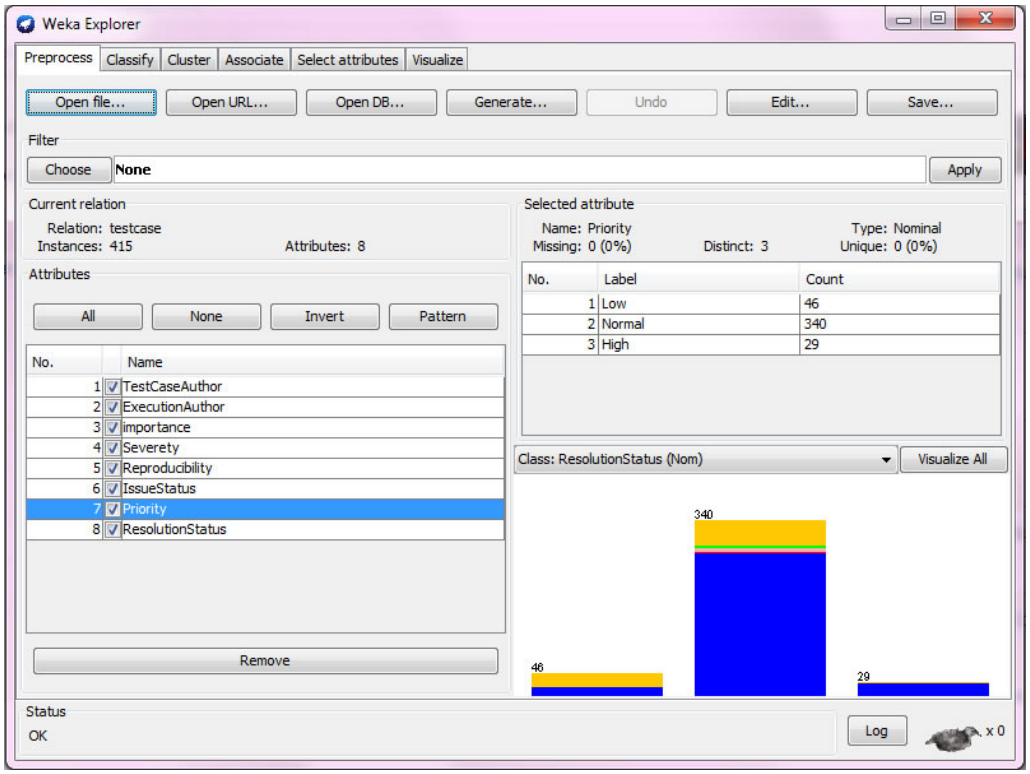

Figure 5.14 - Page Explorer of WEKA

After loading the data set, the Apriori algorithm is executed. Figure 5.15 shows the tab Associate with the selected Apriori algorithm. WEKA returns the most important 10 associations. This number can be changed in the algorithm settings. The listing below shows the results of the associations that were found.

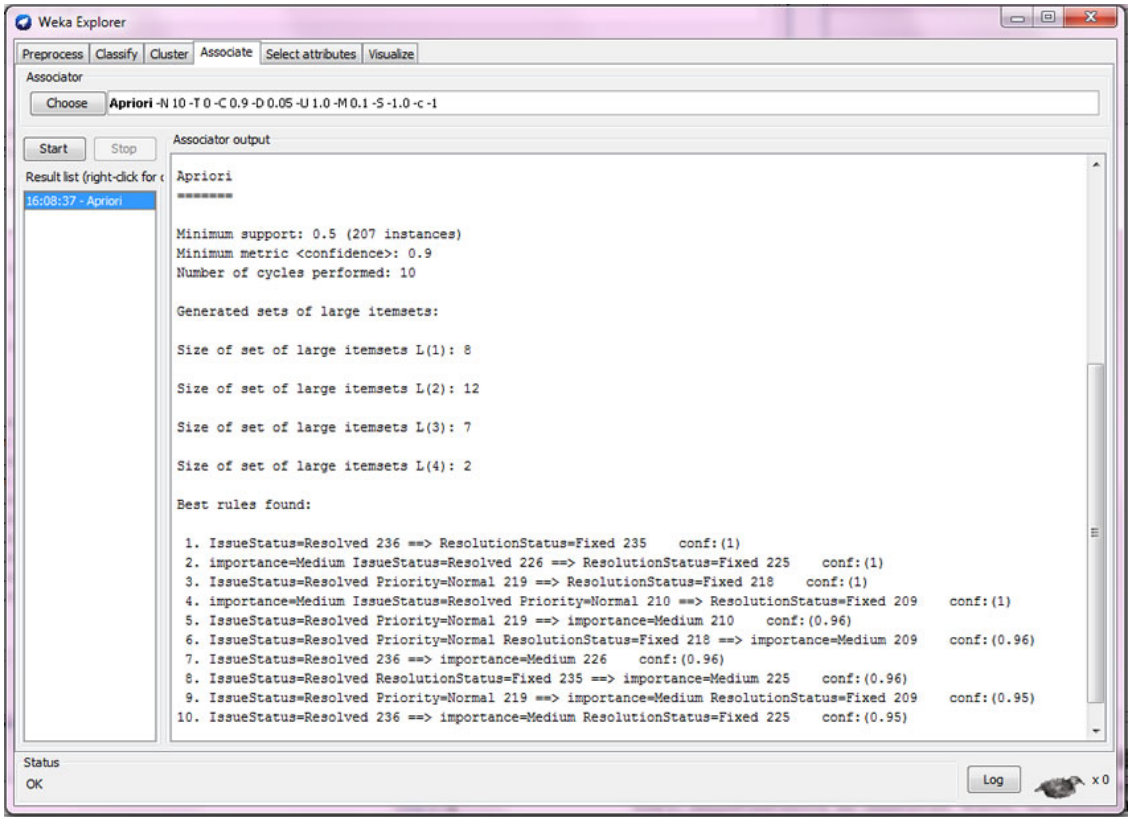

Figure 5.15 - Aba associação com algumas regras geradas

1. IssueStatus=Resolved 236 ==> ResolutionStatus=Fixed 235 conf:(1)
2. importance=Medium IssueStatus=Resolved 226 ==> ResolutionStatus=Fixed 225 conf:(1)
3. IssueStatus=Resolved Priority=Normal 219 ==> ResolutionStatus=Fixed 218 conf:(1)
4. importance=Medium IssueStatus=Resolved Priority=Normal 210 ==> ResolutionStatus=Fixed 209 conf:(1)
5. IssueStatus=Resolved Priority=Normal 219 ==> importance=Medium 210 conf:(0.96)
6. IssueStatus=Resolved Priority=Normal ResolutionStatus=Fixed 218 ==> importance=Medium 209 conf:(0.96)
7. IssueStatus=Resolved 236 ==> importance=Medium 226 conf:(0.96)
8. IssueStatus=Resolved ResolutionStatus=Fixed 235 ==> importance=Medium 225 conf:(0.96)
9. IssueStatus=Resolved Priority=Normal 219 ==> importance=Medium ResolutionStatus=Fixed 209 conf:(0.95)
10. IssueStatus=Resolved 236 ==> importance=Medium ResolutionStatus=Fixed 225 conf:(0.95)

Analyzing the rules some conclusions can be inferred. The fifth rule, for example, shows that out of 219 recorded incidents with status *Resolved* and resolution priority *Normal*, the importance of test cases is *Medium* in 210 of them. This is quite reasonable, because the importance of completing the test case is considered *Medium*,

an incident generated by this test case can also be a priority of correction *Normal*. Just as all other rules, one realizes that there are consistencies among associations that were presented. No irregularity was detected.

If the number of rules to be generated by the settings of the Apriori algorithm is increased, new rules can be analyzed. For demonstration purposes, 50 new rules were generated and some are shown below.

```
...
13. ExecutionAuthor=7 207 ==> TestCaseAuthor=7 201 conf:(0.97)
...
16. Severety=MinorBug IssueStatus=Resolved ResolutionStatus=Fixed 193 ==> importance=Medium 187 conf:(0.97)
...
29. TestCaseAuthor=7 ExecutionAuthor=7 201 ==> ResolutionStatus=Fixed 190 conf:(0.95)
...
36. Severety=MinorBug 298 ==> Priority=Normal 277 conf:(0.93)
...
48. Reproducibility=Always Priority=Normal 209 ==> importance=Medium 191 conf:(0.91) ...
```

For the registration of a knowledge item of the mined item type in Testing KM Portal, generic information about that item were considered given the diversity of methods and algorithms that exist in data mining. In Figure 5.13, the *MinedItem* table shows the attributes that are available for the registration of a mined item. The attributes are: Description, Algorithm, Configuration, Result and Analysis.

5.3.3 Loading Existing Knowledge Items

From the conceptual model of Testing KM Portal shown in Figures 5.12 and 5.13, the knowledge repository for the system was created. Knowledge items already existing in the organization must be loaded to the repository.

In the context of the ICAMMH Project, there were several test cases available stored in MantisBT and TestLink. However, each one of these tools has its own data repository, implemented in different ways, demanding a thorough analysis of the structure of each one in order to load the data. Moreover, each tool has its own terminology to represent the manipulated data, i.e., different terms are used to represent the same concept. Thus, to load existing test cases, a functionality was developed to connect and get data from the repositories of MantisBT and TestLink, and then convert them into objects (instances) of the data schema of Testing KM Portal. This procedure is illustrated in Figure 5.16. From the ICAMMH Project, 1568 test cases were loaded. Details of this step can be viewed in Appendix B.

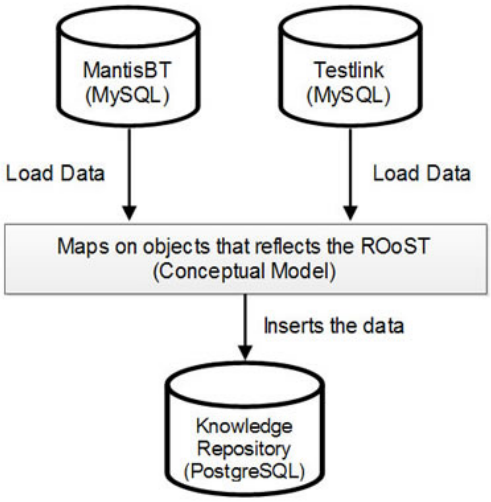

Figure 5.16 - Loading existing knowledge items

5.4 Testing Knowledge Management Portal (TKMP)

The Testing KM Portal produced by the specification presented above is presented in this section. The KMS prototype of software testing, called TKMP, is part of an extension to the one proposed in Coelho (2010) and an ongoing project that implements in (SPECIMILLE et al., 2014). Thus, details on the development of TKMP can be found at (SPECIMILLE et al., 2014).

Figure 5.17 presents the main page of TKMP. Testing KM Portal recognizes the authenticated user profile in the system, detecting whether the user is a knowledge manager or a developer. The features of Testing KM Portal are set according to the user profile. As shown in the top of the central panel of Figure 5.17, the current authenticated user is *Erica F. Souza* and her profile is *Knowledge Manager*.

As seen in Figure 5.17, TKMP is divided into several panels:

- **Functionalities:** contains the main portal functionalities, which are creation and search for knowledge items.
- **Tools to Support Collaboration:** incorporates tools for supporting collaboration, namely Yellow Pages and Discussion Forums.
- **Your Participation in Portal:** contains information about current authenticated user´s participation in the portal. From this panel, the user can

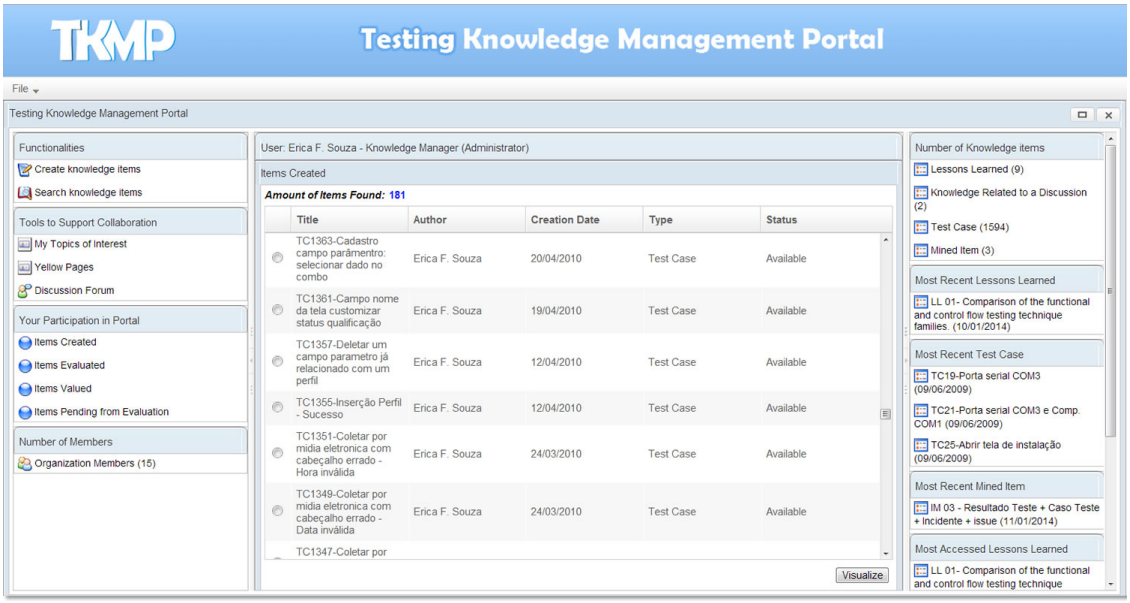

Figure 5.17 - Main page of TKMP

view the knowledge items created, evaluated, rated and to be evaluated by it.

- **Number of Members:** reports the number of active members in the organization. From this panel, the user can view a list of all the organization members.

- **Central Panel:** it is the panel located in the central region of the page. It displays information related to the portal functions being performed.

- **Number of Knowledge Items**: informs the amount of existing knowledge items per type.

- **Most Recent Items:** presents the knowledge items that were more recently created.

- **Most Accessed Items:** presents the most accessed knowledge items in the portal.

The following subsections illustrate the use of TKMP in supporting activities of knowledge items creation, evaluation, search, valuation and maintenance.

5.4.1 Knowledge Item Creation

In functionalities panel, there is an option to create knowledge items. After selecting this option, the Types of Knowledge Item are displayed in the central panel (Figure 5.18). The specialist can choose one of the following knowledge items: *Lessons Learned, Knowledge Related to a Discussion, Test Case* or *Mined Item.*

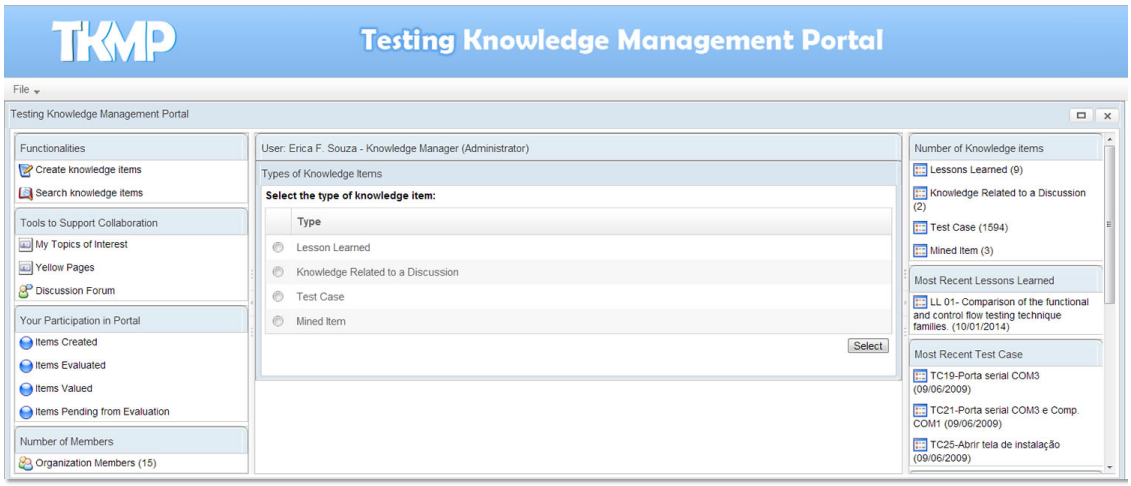

Figure 5.18 - Types of Knowledge Items

When a type of knowledge item is selected, a registration window is displayed. Tabs are used to group two sets of information about the knowledge item that must be filled: General Information and Specific Information. The fields contained in the General Information tab are the same for any type of knowledge item. General Information considered are: Author, Creation Date, Title, Summary, Related Project and Applicability. On the other hand, the tab Specific Information contains specific information about the knowledge item to be created.

Figure 5.19, presents creating a knowledge item of type of *Test Case*. The tabs General Information and Specific Information are highlighted.

When selecting Specific Information about the test case, it is divided into two groups (subTab), presented in Figure 5.20. *Test Case Data* presents the information necessary to create a test case to be registered; *Test Case Result* presents all pertinent results information of executing test case.

- **Test Case Data**: Created by, Test Level, Testing Technique, Importance, Input Data, Expected Result, Steps, Test Code, Code to be Tested.

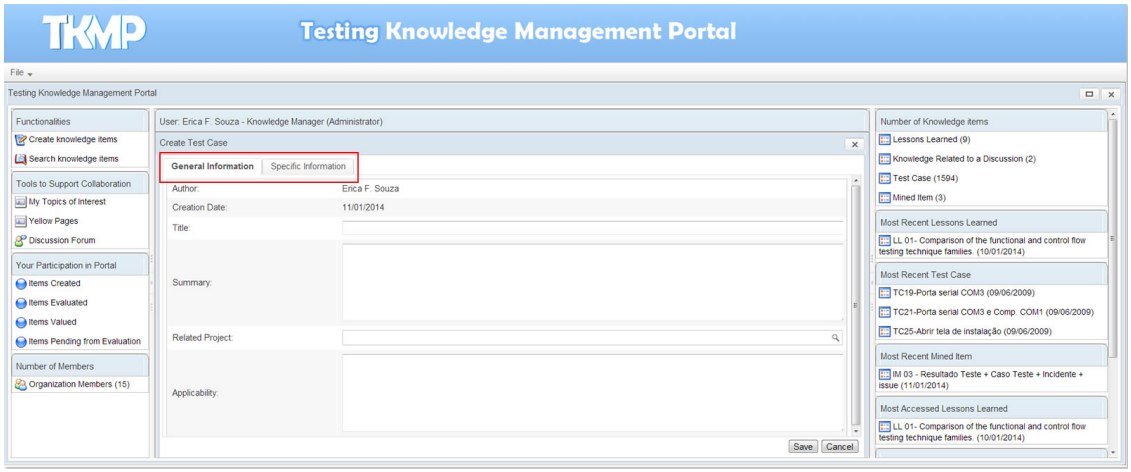

Figure 5.19 - Creating Knowledge Item - *Test Case*

- **Test Case Result**: Tester, Execution Status, Tested Product Version, Actual Result, Description, Execution Status. If execution status equals "Failed", then the following information must be filled: Summary, Description, Original Version of the Product, Generated Version of the Product, Severity, Reproducibility, Resolution Status, Issue (Submission Date, Priority, Issue Status).

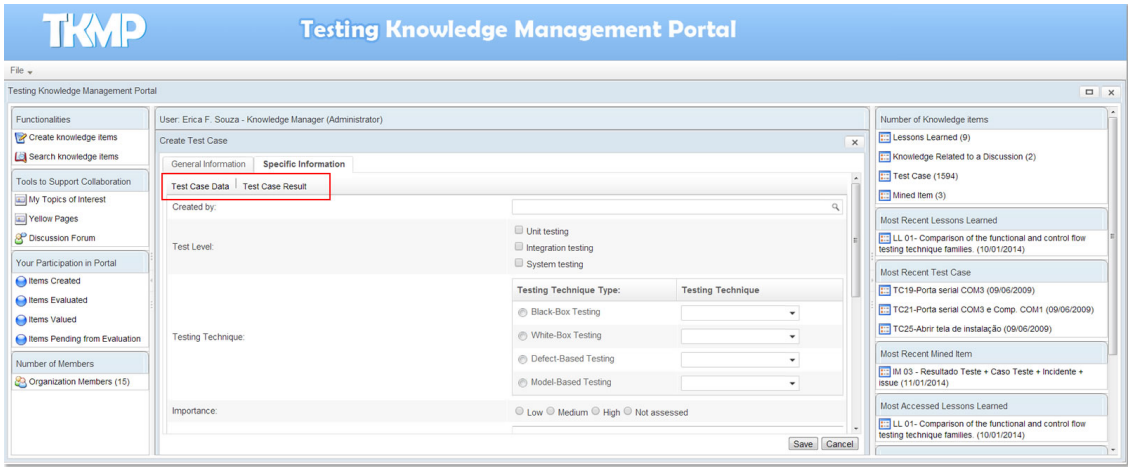

Figure 5.20 - Create Test Case - Specific Information

With respect to other types of knowledge items, the specific information presented are:

- **Lessons Learned:** Type (Success, Failure, Informative), Problem Descrip-

tion, Solution Adopted or Recommended, Expected Result.

- **Knowledge Related to a Discussion**: Knowledge Acquired, Strengths, Weaknesses, Link to the discussion.
- **Mined Item**: Create by, Algorithm, Configuration, Description, Result, Analysis.

5.4.2 Knowledge Items Evaluation

The knowledge item must be evaluated by the knowledge manager before being made available to organization members. Through option *Evaluation Pending Items Available*, the knowledge manager can view the knowledge items created that need to be evaluated. After selecting this option, the *Evaluation Pending Items* appears in the central panel, as shown in Figure 5.21.

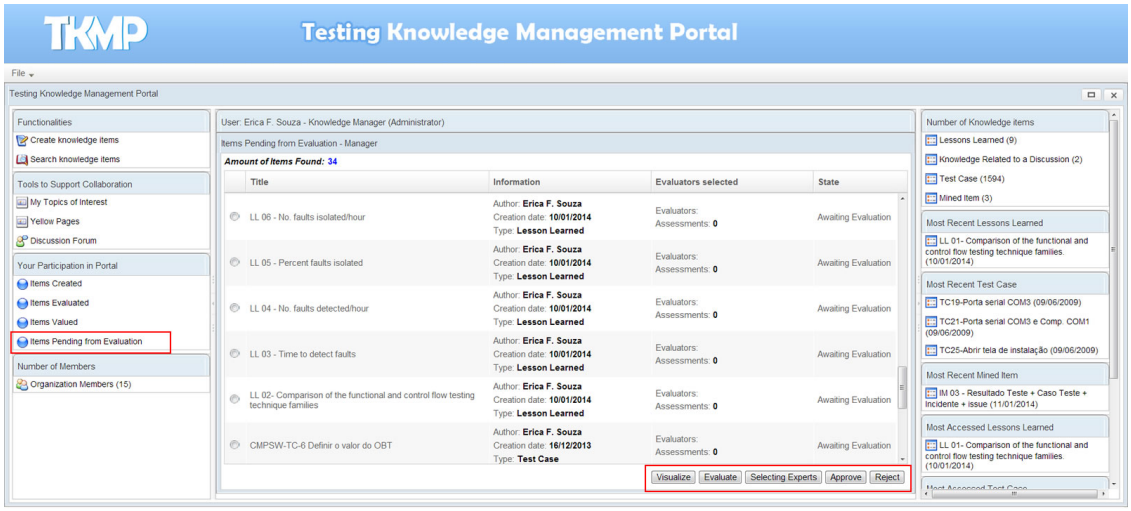

Figure 5.21 - Items Pending from Evaluation

The knowledge manager can select a knowledge item pending to evaluation, visualize their information, change them, evaluate and approve it or reject it. Moreover, the knowledge manager can request that the item be evaluated by an expert, selected with the support of the Yellow Pages system. Thus, any organization member can be selected to evaluate a knowledge item.

When an expert is selected, the knowledge item becomes available to be evaluated by him/her. When authenticated to the portal, the expert can select the *Items Pending from Evaluation* option to check the items that must be evaluated. However, the

Pending Items Evaluation window is dynamic and is displayed according to the authenticated user profile. When the user is a common organization member, it is only possible to view the information of the knowledge item and evaluate it (see Figure 5.22). Only the knowledge manager is allowed to select an expert to review, approve or reject knowledge items.

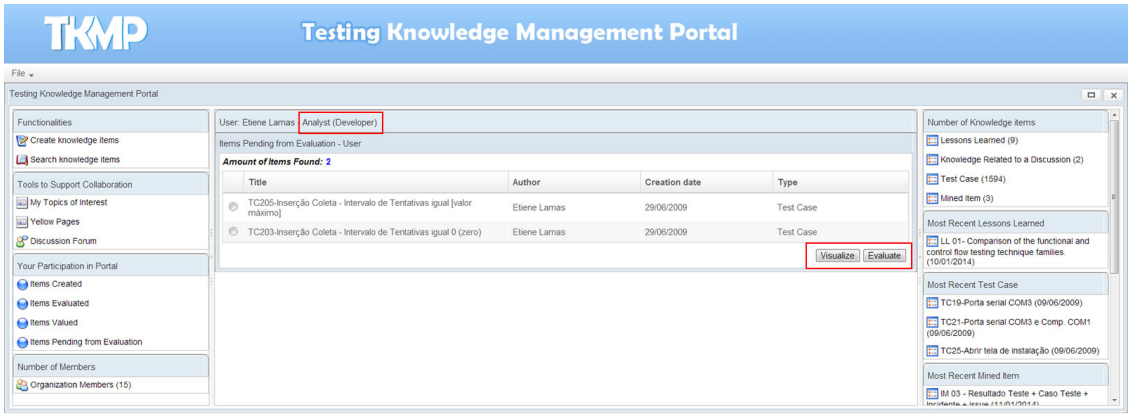

Figure 5.22 - Items Pending from Evaluation (Common User)

When knowledge item is selected by the expert to carry out the evaluation, the *Evaluate Knowledge Item* window appears, as shown in Figure 5.23. The expert shall inform the score of each evaluation criterion defined, Feedback and Final Results of evaluation (Approved, Approved with modifications, Not approved or Indefinite) and, if necessary, can describe an opinion on evaluation. In this thesis, as well as in ODE´s KM Portal (COELHO, 2010), the criteria suggested in Montoni (2003) are indicated for the evaluation of knowledge items, namely: Correction, Completeness, Consistency, Usefulness and Applicability.

After an expert finalizes the evaluation of knowledge item, the knowledge manager can preview it from *Pending Items Evaluation* window. So, after selecting an item and select to view their information, the *Knowledge Item View* window appears in the central panel, containing all the information of the knowledge item, including its evaluations. In *Evaluation* tab, the knowledge manager can view all the evaluations made by expert, as shown in Figure 5.24.

After viewing the evaluation performed by the experts, the knowledge manager can decide whether to approve or reject the knowledge item. This decision may be taken independently of the evaluations made by expert. That is, the knowledge manager has the autonomy to decide whether the item should be approved or rejected. After

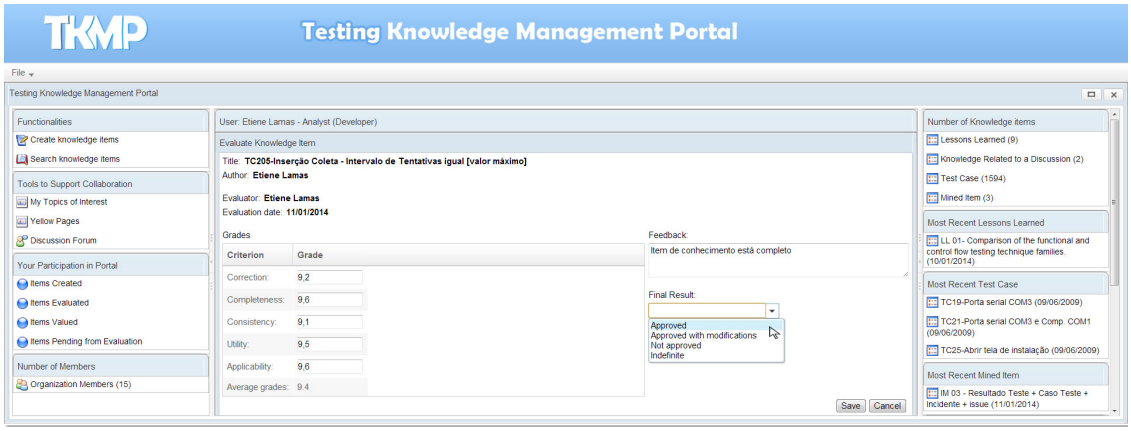

Figure 5.23 - Evaluating Knowledge Item

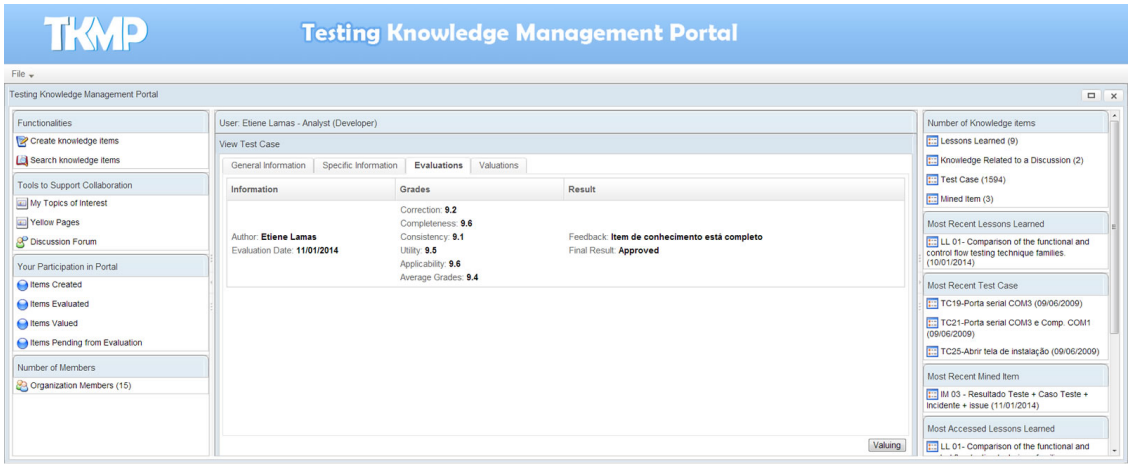

Figure 5.24 - Knowledge Item View - Evaluation

approval of the knowledge item, it becomes available to the organization and can be accessed and used by members. The status of each knowledge item can take the following values:

- Awaiting Evaluation: Indicates that the knowledge item is being evaluated.
- Approved: Indicates that the knowledge item has been approved and is now available to be accessed by organization members.
- Rejected: It points that the knowledge item was rejected and not available to organization members.

5.4.3 Knowledge Item Search

In functionalities panel, there is the option to *Search Knowledge Items*. With this option, the search criteria are displayed in the central panel (Figure 5.25). The specialist can search by the general criteria involving all types of knowledge items. However, when selecting the knowledge item of type of *Test Case*, a set of specific criteria for test case are presented. Table 5.5 presents the criteria used.

Table 5.5 - Search Criteria

General Search Criteria	Search Criteria for Test Case
Search Criteria	Test Level
Creation Date	Testing Technique Type
Last Access	Testing Technique
Number of Accesses	Importance
Number of Valuations	Execution Date
Percentage of Positive Valuations	Execution Status
Percentage of Negative Valuations	Severity
Related Projects	Reproducibility
Related Themes	Resolution Status
Knowledge Item Type	Priority
	Issue Status

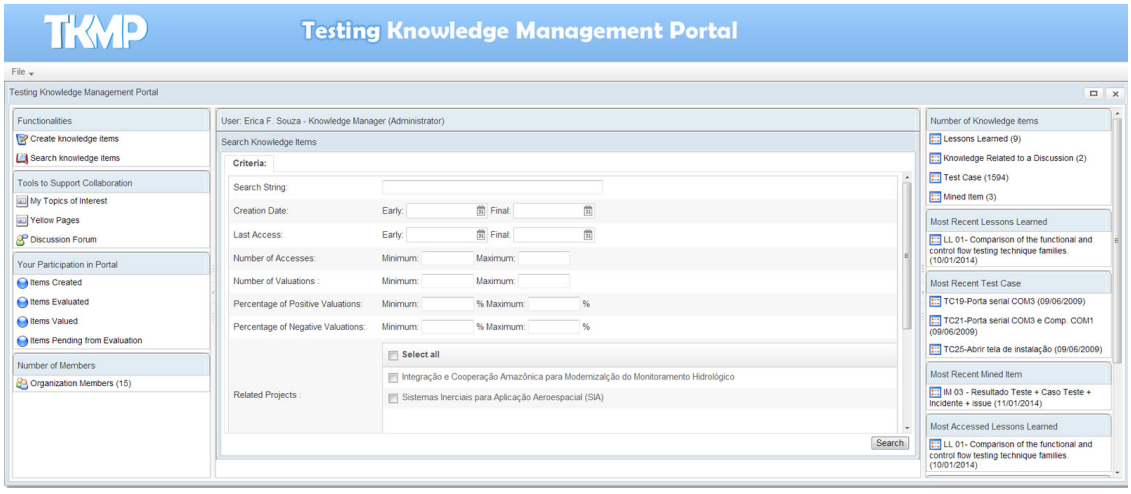

Figure 5.25 - Search Knowledge Items - Search Criteria

The search criteria can also be combined. After informing the parameter settings

of search and to execute, a list of found knowledge items is displayed. A search example is shown in Figure 5.26. In this search example, the following criteria were informed: knowledge items related to the ICAMMH Project, *Test Case* type, with a *High* importance level and execution status is *'failed'*. Test cases that are of high importance and that failed can provide an important learning experience for the organization and avoid the same mistakes in future projects. This search returned 27 knowledge items.

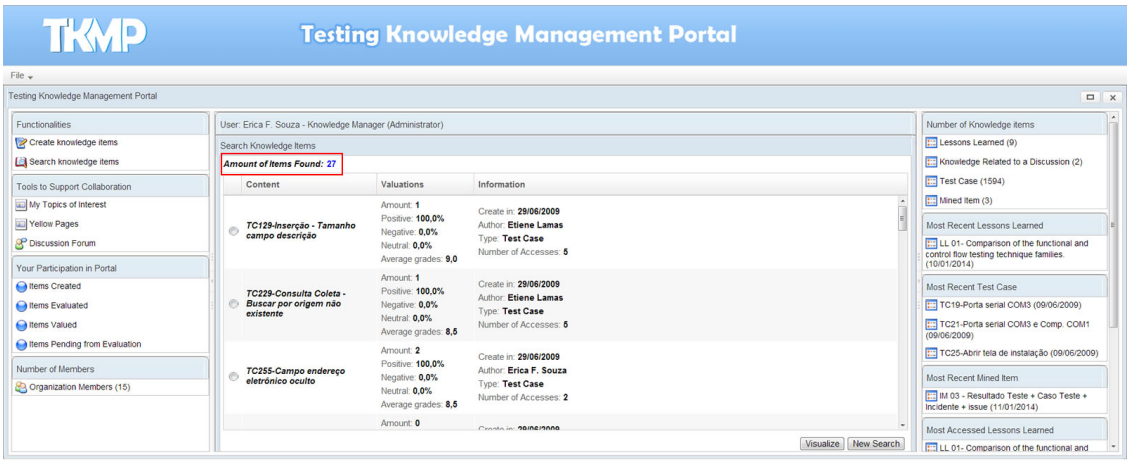

Figure 5.26 - Test Cases returned

The list of knowledge items returned through search (Figure 5.26) contains a summary of information for each item found, among them a content summary of the knowledge item, a summary of their evaluations, author, creation date, type of knowledge item and number of accesses. To view all the details of a knowledge item simply select it and all information can be viewed in *View Item Knowledge* window, as shown in Figure 5.27.

In relation to a test case, entire history of this test case is returned. This will benefit the traceability of the test case. The following specific data for the test case "*TC1341-Inserção Coleta - Falha de Intervalo de Tentativas * Falha Max de Tentativas > Periodicidade*" are presented. In Figure 5.28, the specific information about the test case *TC1341*, and its results are presented. Note that this test case was exercised three times. So, three results will be displayed separately. Figure 5.29 shows these results.

When a test result has the execution status as 'failed', the incident will be presented along with its respective result (Figure 5.30).

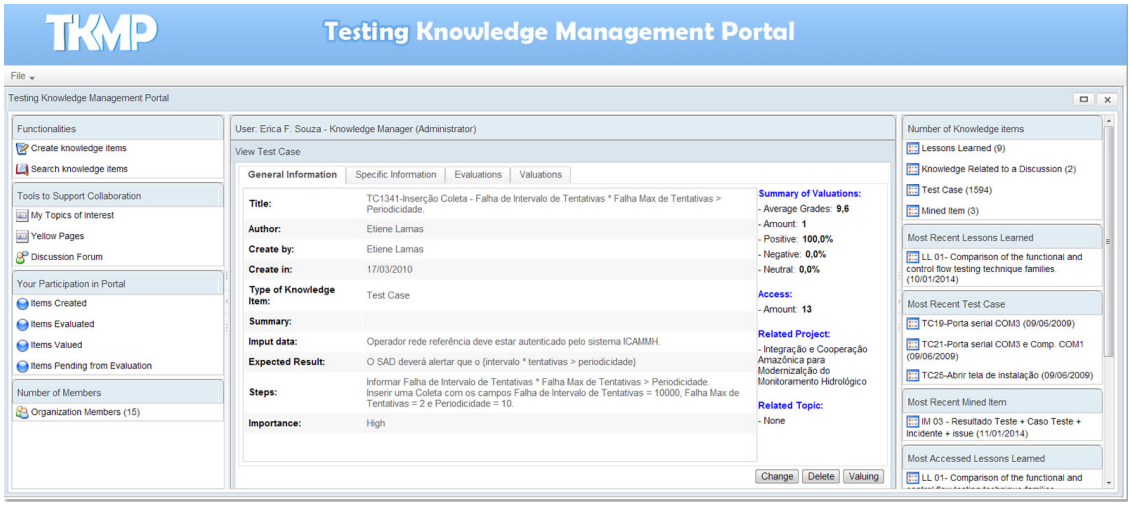

Figure 5.27 - View Test Case returned

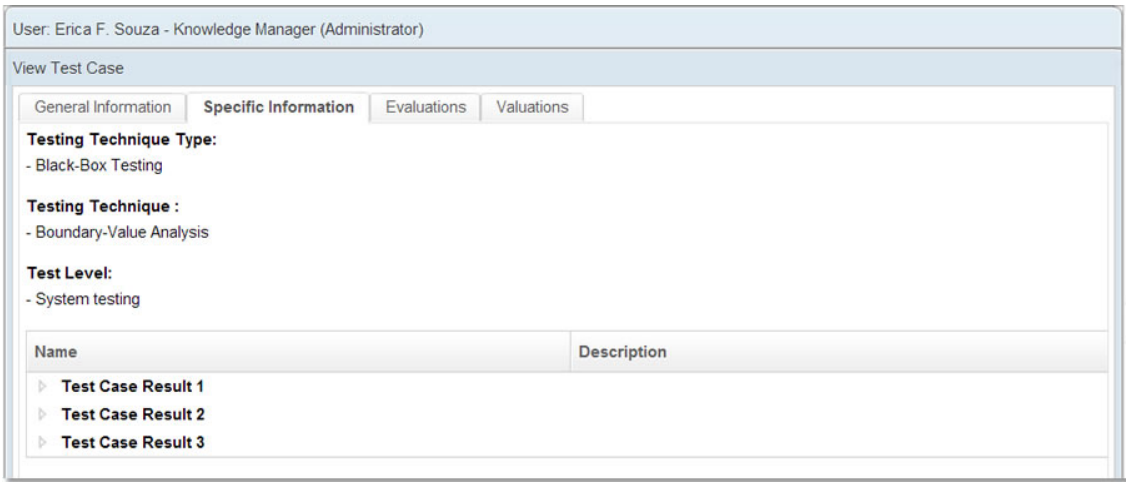

Figure 5.28 - View Test Case returned - Specific Information

All information, including evaluations and valuations made about the knowledge item can be visualized. The valuations are useful because they allow the organization member to consult a knowledge item and also view the opinion of other members on the usefulness of this item. Figure 5.31 shows the *View Item Knowledge* window open with all the valuations made about the knowledge item Valuations tab.

5.4.4 Knowledge Items Valuation

View Test Case window allows to perform new valuations of the item triggering the Valuing button on the bottom right of the window (see Figure 5.31). The *Valuing Knowledge Item* window appears, as shown in Figure 5.32. In this window, a degree of

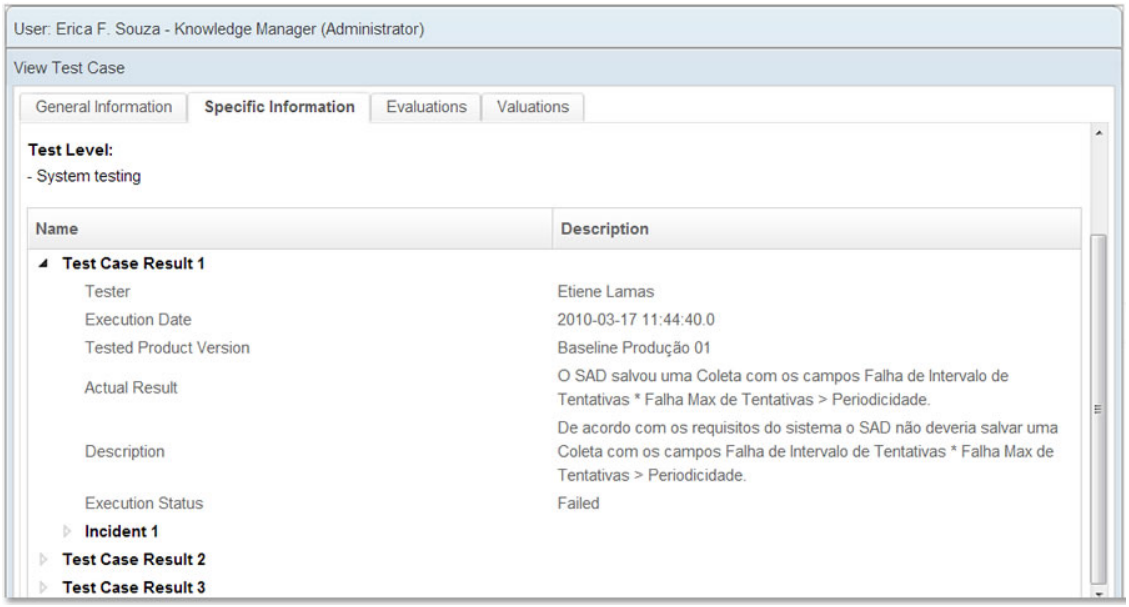

Figure 5.29 - View Test Case returned - Test Case Result

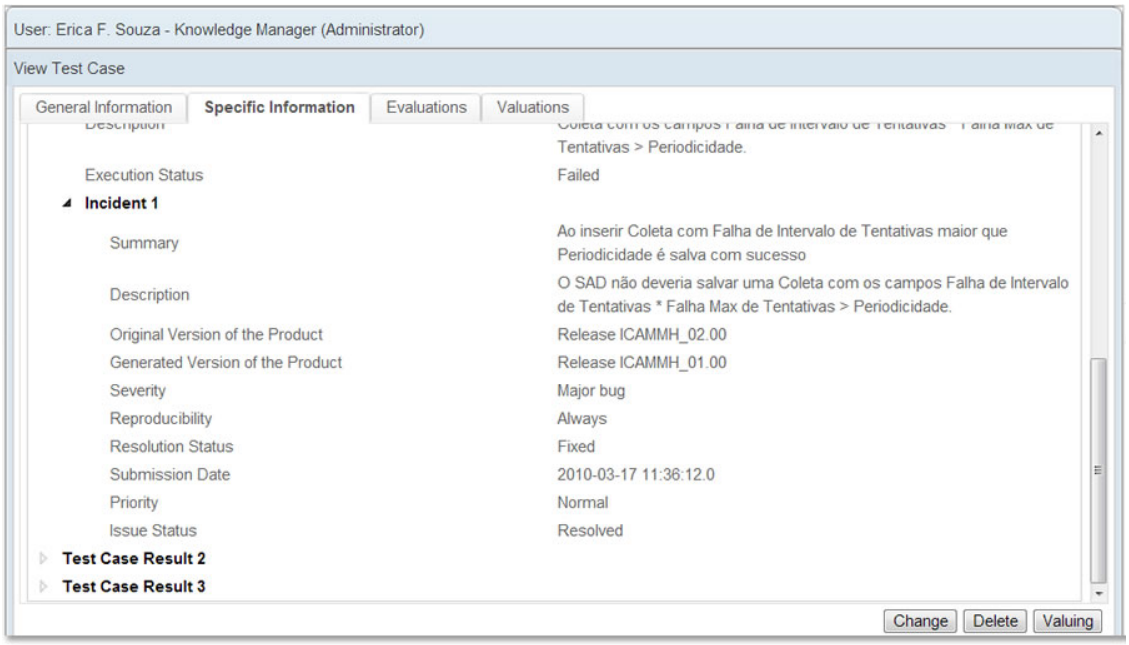

Figure 5.30 - View Test Case returned - Incident

utility in value from -10 to 10 must be informed. Values below zero indicate that the valuation is negative, values above zero indicate that the valuation was positive and a value of zero states that the valuation is neutral. Besides the degree of usefulness, a comment on the evaluation can be done. After confirming the valuation, it is available to be viewed alongside the other valuations made about the knowledge

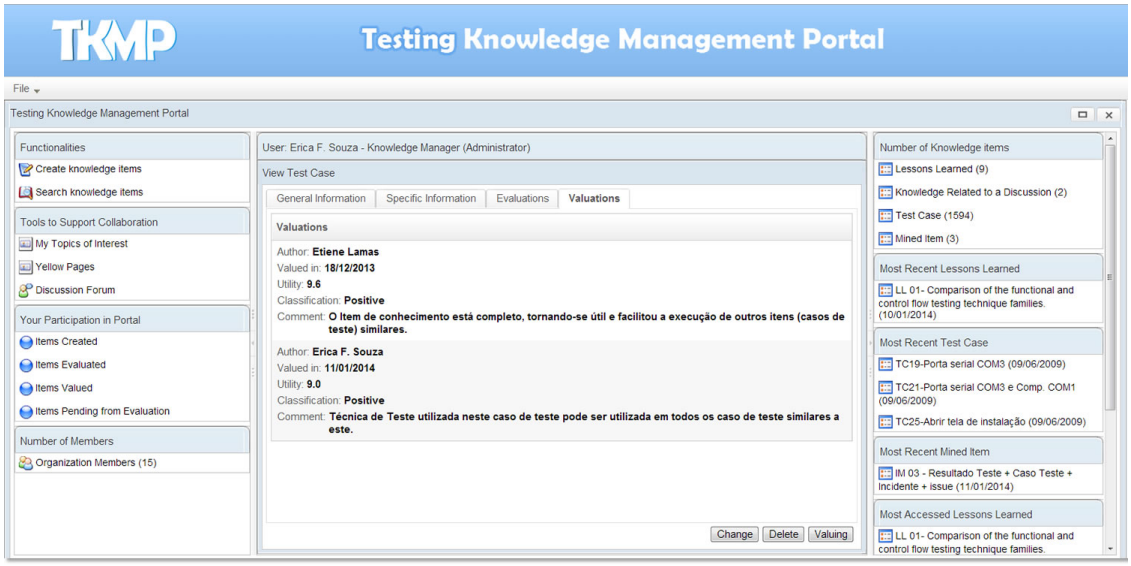

Figure 5.31 - View Test Case returned - Valuation tab

item.

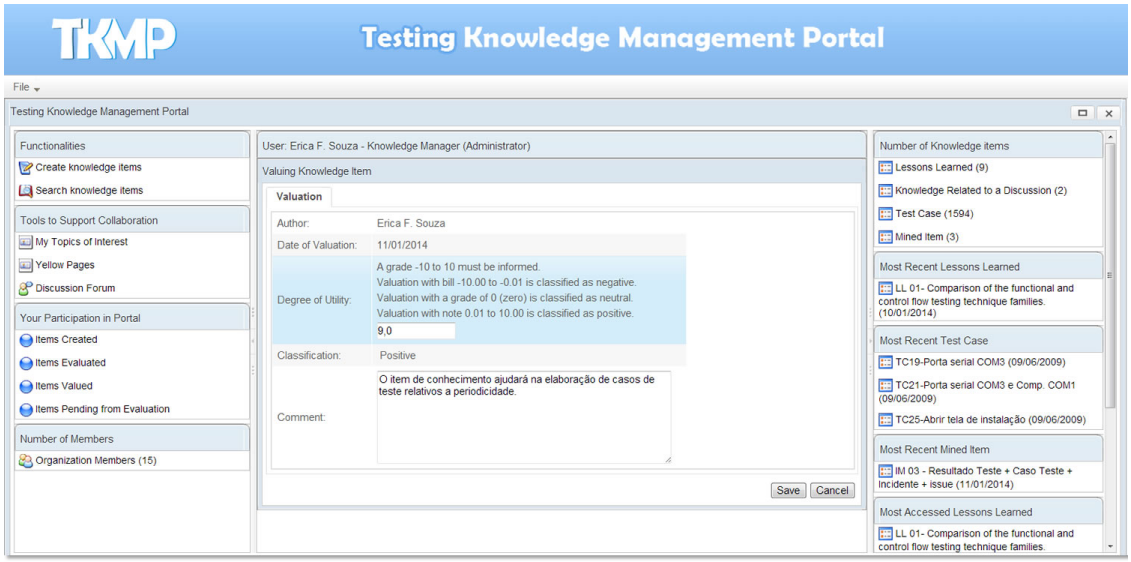

Figure 5.32 - Valuing Knowledge Items

5.4.5 Knowledge Items Maintenance

In TKMP, maintenance is done only by a user with knowledge manager profile. To find candidate items to be updated or deleted, the manager must search, by the search engine, the knowledge items to perform maintenance. Thus, knowledge

items are presented to the knowledge manager that in turn selects one to view its information, as shown in Figure 5.33. *View Item Knowledge* window contains the Change and Delete buttons that allow, respectively, the change and deleting the displayed item.

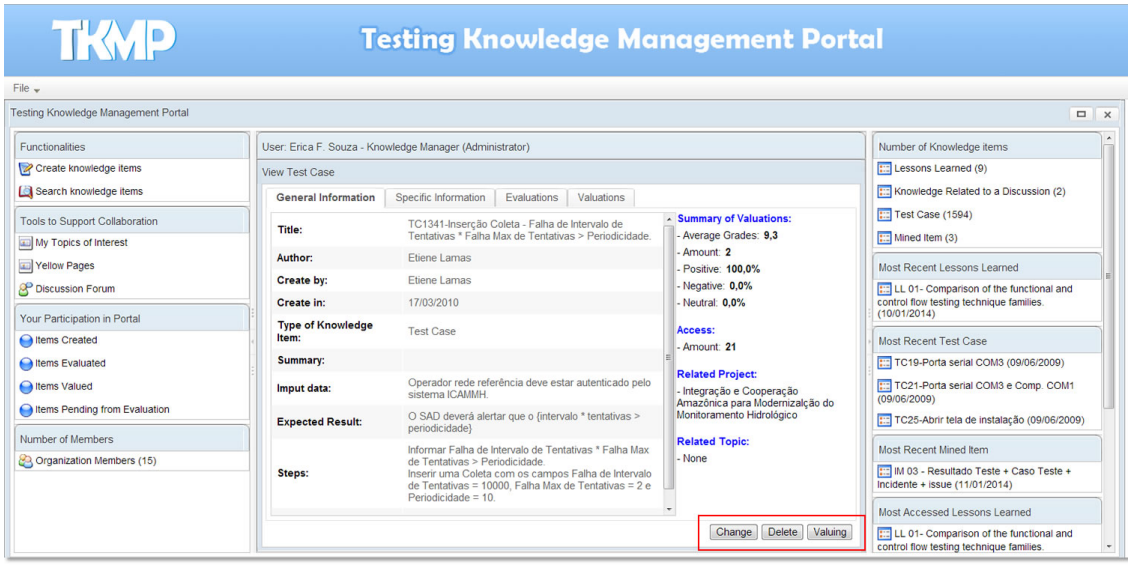

Figure 5.33 - Maintenance of Knowledge Items

5.4.6 Yellow Pages

In the Yellow Pages integrated with TKMP (see Figure 5.34) it is possible to inform the profile of those to be found in the portal. That is, one can find people according to their skills, projects they participated, activities they worked, a position held and topics of interest. After entering the desired profile, people are found listed along with contact information for each. Also in this list, one can select a person and view all the information in *Your Profile View* (Figure 5.35) window.

In View on Your Profile window, it is possible to identify, besides the name and contact, the competences of each, the projects and activities involved, the functions performed, discussions one participated, topics of interest, besides the items created, evaluated and valued.

The Yellow Pages system can be used to find people profiles to solve specific problems or perform specific tasks, in particular, the evaluation of knowledge items. Thus, to evaluate a knowledge item, the knowledge manager can select people using this system.

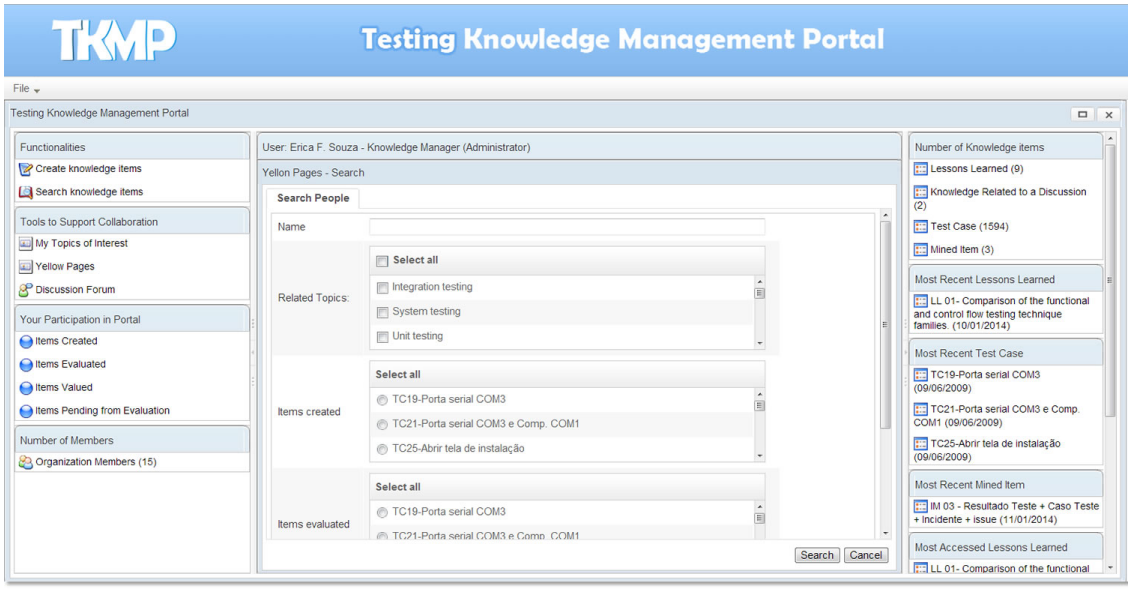

Figure 5.34 - Yellow Pages window

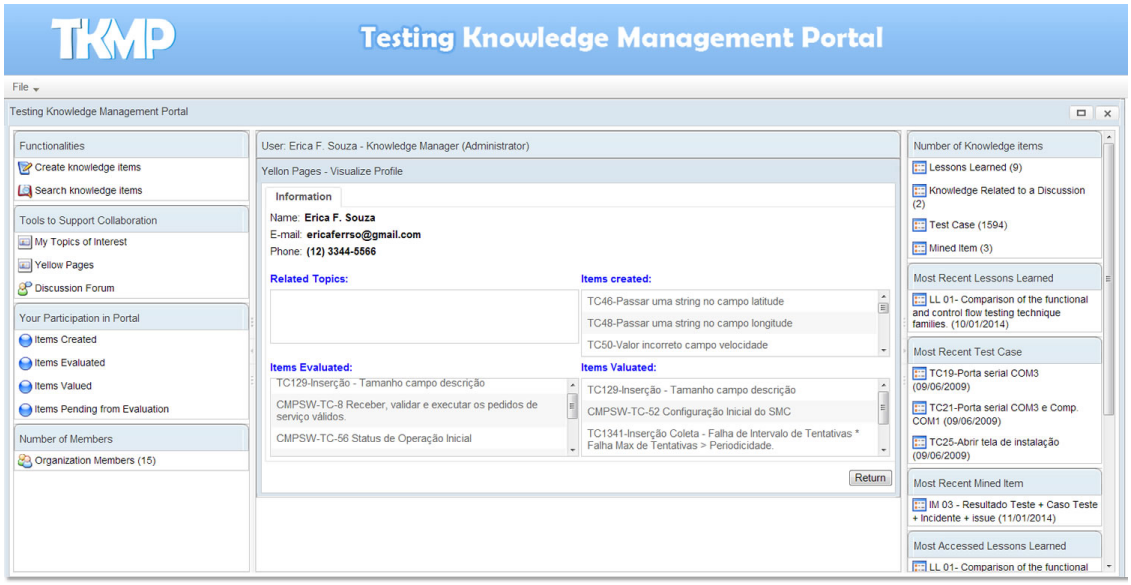

Figure 5.35 - Visualize Profile

5.4.7 Discussion Forums

Testing KM Portal allows the creation of knowledge items related to discussions. The procedure to create an item of this type is the same as presented in Section 5.4.1, which deals with the creation of knowledge items in the portal. The only difference in the process are the specific information to be filled, which in this case are of a specific knowledge item concerning a discussion. Specific information on this type of

knowledge item, is necessary inform the acquired knowledge of the discussion, the strengths and weaknesses found in addition to the link for the complete discussion. Figure 5.36 shows the window *Create Knowledge Relating to a Discussion*.

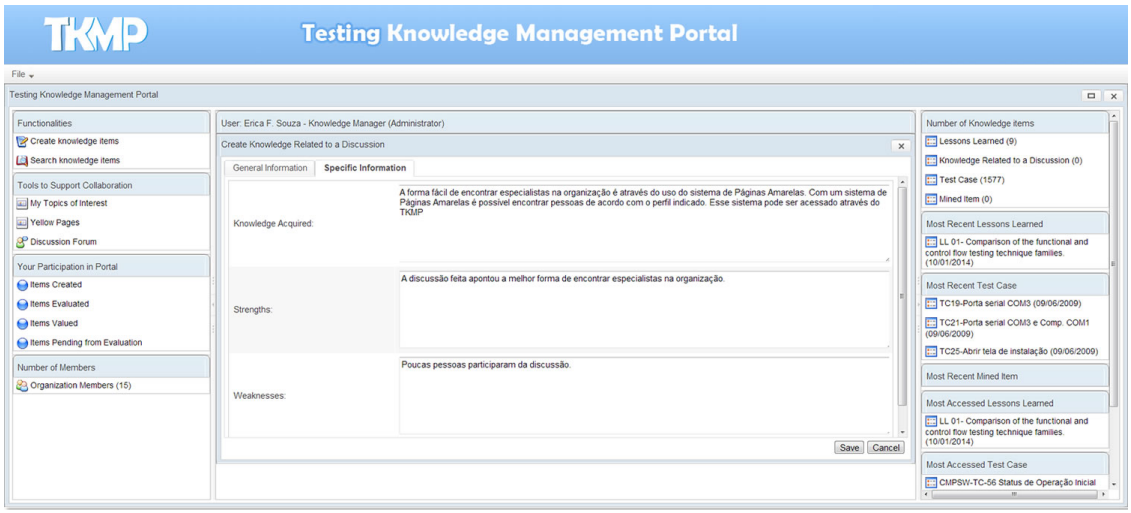

Figure 5.36 - Maintenance of Knowledge Items

Discussion forums are an effective means of communication and share tacit knowledge among organization members, besides being a source for creating new knowledge items and a tool to support group work. Thus, it was decided to integrate some discussion environment to Testing KM Portal forums.

An integration process is in progress from the *JForum*[2] environment. The *JForum* allows to instantiate a custom discussion forums environment for an organization providing functionalities, such as: registration of discussion forums, membership of topics, as well as user registration and their participation in the topics available.

5.5 Evaluation

Testing KM Portal was evaluated by leaders of two different projects. One was a sub-project leader of ICAMMH and the other is one of the project coordinators of On-Board Data Handling (OBDH) Software, inside Inertial Systems for Aerospace Application (SIA) Project. It is important to note that the evaluation was quite preliminary as ICAMMH was in the end and the testing activities of OBDH software are in the initial phase. However, meetings, with the leaders of both the projects by means of presentations and system demonstrations, lead to improve the Portal.

[2]http://jforum.net/

Both the leaders stressed the importance of such a system to benefit Software Testing Processes, in particular, to critical systems such as developed at INPE and DCTA.

With respect to ICAMMH, it was observed that the Portal would be extremely valid for the project development. As the project dealt with a significant number of resources (both human and others), it was observed that there was a loss of knowledge due to the turnover rate of the team members. Therefore, according to the leader of ICAMMH, a Portal such as Testing KM, would be definitely beneficial in tracking, for example, similar test cases, and project them to other similar situations in different modules and also in future projects.

Due to time limitations, it would be impractical to adapt the Portal to second case study, OBDH, especially as the testing activities of OBDH software are in the beginning. Moreover, as SIA deals with critical systems, there is an issue of confidentiality and other factors. Therefore, the initial idea was to use what was already done for ICAMMH to SIA. Interestingly, most of the situations presented by SIA were completely compatible with what was already developed for ICAMMH. Naturally, there would be a point that more extensions or a specific adaptation may have to be developed depending on the particular specificity of SIA.

The second project analyzed is the first version of the OBDH software for the on-board computer of Project SIA. This project aims, among others, the development of a computational system for OBDH an Attitude and Orbit Control (AOC) of satellites that can be adapted for future space applications at INPE. The first version of the OBDH software is in testing phase at the moment. The final version of this software will add all the functionalities of OBDH of the satellite. Its main functionalities are: (i) receiving and analyzing ground station telecommands; (ii) Formatting and transmission of telemetry; (iii) Data acquisition from on-board subsystems (Real Time and Stored); (iv) Housekeeping; and (v) Fault Detection Isolation and Recovery (FDIR).

With respect to OBDH software, the leader´s evaluation about the Portal was that such a Portal would be very important while dealing with critical systems. However, he pointed out that the main issue is that the organizations that deal with Software Testing must be open minded in order to change their culture and be ready not only to accept new concepts and tools but also to implement such new ideas. Since it is the development of critical systems, organizations generally have a more conservative approach and adopt new methodologies/tools is made more slowly than in an environment of commercial software development.

5.6 Final remarks about this chapter

In this chapter the T-KM was employed to build a KMS for managing software testing knowledge, as a proof of concept of the feasibility of applying the proposed framework. The prototype was developed in the context of ICAMMH project. The data used as knowledge items of Test Case type corresponding to the data stored in the repositories of MantisBT and TestLink tools. A second project also was analyzed, the first version of the OBDH software inside SIA Project. Finally, Testing KM Portal created was evaluated by leaders of projets ICAMMH and OBDH software.

In the next chapter are presented: the main considerations of this research, the main proposed solution contributions, the main limitations and difficulties encountered, suggestions for future work, and the final considerations.

6 CONCLUSIONS

This chapter presents the main conclusions of this work. The following sections present: the main considerations of this research, its main contributions, the main limitations and difficulties encountered, suggestions for future work, and the final considerations.

6.1 General considerations

This work defines a Knowledge Management (KM) framework, called T-KM, to manage software testing knowledge, so that different testing knowledge items are collected, shared, reused and improved throughout the organization. T-KM is an ontology-based framework for guiding KM initiatives in the software testing domain, supported by a Knowledge Management System (KMS).

KM in software testing has shown to be a very promising research area, since it KM helps in handling knowledge within the organization in several respects, as shown by a systematic mapping conducted (Section 2.3.1). However, the mapping also showed that KM in software testing still seems to be a challenge, mainly on how to represent knowledge. This can be achieved by representing the information using, for example, ontologies. Although recognized as an important instrument by the KM community (BENJAMINS et al., 1998; KIM, 2000; STAAB et al., 2001), ontologies are not being widely used in KM initiatives in software testing. Thus, assuming that ontology-based KM applied to software testing is an important and a promising research line, a SLR was conducted in order to investigate ontologies in the software testing domain. 12 ontologies addressing the software testing domain were identified. These ontologies have several limitations. Therefore, a Reference Ontology on Software Testing (ROoST) (SOUZA et al., 2013b) was developed, to contemplate the necessary characteristics of a "beautiful ontologies" (D'AQUIN; GANGEMI, 2011) and meet the objectives of this thesis.

Once established an ontology for software testing, a process to guide organizations in the accomplishment of KM in software testing was also developed. Thus a framework was generated, consisting of two main components: ROoST and a process with a set of guidelines for implementing the KM in software testing organizations. As a proof of concept of the framework, a prototype of a Testing KM Portal has been developed under the T-KM framework to evaluate the framework. Moreover, data from an existing project were used and Testing KM Portal was also evaluated by project leaders from two actual scenario.

Based on the results presented, it is believed that the T-KM framework constitutes a guide to support the implementation of KM initiatives in the software testing domain. The support of a KMS in software testing can guide team members test the reuse of knowledge items, such as: Test Cases, Lessons Learned, Knowledge Regarding Discussions and Mined Items. Thus, the implementation of a KMS can support learning through the sharing of knowledge acquired when there is available knowledge, because this facilitates the sharing and use of organizational knowledge, so that members of the organization are comfortable in interacting with KM in the organization. Moreover, a KMS can also support the management of skills and abilities of the organization members, because people best placed to make a specific decision or solve a particular problem can be found.

6.2 Contributions

Although the knowledge in software testing is necessary, a systematic search of the literature (Section 2.3.1) shows that solutions that exist are not enough to efficiently perform all testing activities, when it comes to KM within the organization. Thus, the main contribution of this thesis is a KM framework to manage software testing knowledge. This framework (T-KM) provides guidance and direction on how KM in software testing should be carried out in the organization.

Besides the main contribution that is the framework, each of its their components featuring a particular contribution, since they are used and evolved in the context of the framework as a whole, can also be used and evolved independently. So, the following contributions are also in order:

(a) A Reference Ontology on Software Testing (ROoST). The KM community recognizes ontologies as an important instrument for representing KM (BENJAMINS et al., 1998; MAEDCHE; VOLZ, 2001; AHMAD et al., 2011; VALASKI et al., 2012). ROoST helps to address issues related with the application effectively of KM in software testing so that knowledge items can be shared and reused. For properly managing testing knowledge, it is necessary to have a common understanding of the testing concepts, in order to associate semantics to a large volume of test information. ROoST helps to minimize this problem. Further, ROoST includes many important characteristics with regard to "beautiful ontologies" (D'AQUIN; GANGEMI, 2011).

(b) A process defines a set of fundamental principles for implementing KM in

Software Testing. This process provides directions on how to identify KM goals that are specific for a particular software testing organization, and how to achieve a Knowledge Management System (KMS) to support the organization's KM initiative in software testing.

Besides the contributions directly related to the framework, additional contributions were also achieved:

(a) Making evident some aspects associated to the employment of KM in software testing through a mapping study. This mapping study, among other aspects investigated, provided an overview of the main problems reported by organizations related to the lack of knowledge about software testing as well as the actual benefits of the current studies of KM in software testing. These results show areas for future research and provides a map that allows appropriately to position new research activities.

(b) Making evident how widespread are the use of ontologies for managing software testing, and the characteristics of such ontologies through a Systematic Literature Review (SLR). 12 different ontologies were found applied in various aspects of the domain of software testing. Although these ontologies have several limitations and considered inappropriate for purposes of this thesis, this SLR showed different contexts in that these ontologies are applied and that could guide new investigations besides KM. Furthermore, the realization of this SLR motivated the creation of ROoST.

(c) Partial implementation of ROoST in a language epistemological level (OWL). One advantage of using ontology in OWL is that it can promote other research possibilities on ROoST in a computational ontology language.

(d) Survey to identify a specific scenario the software testing domain to exercise the KM of most interest for experts in the software testing field

(e) Requirements definition and development of a Testing KM Portal. These requirements help to specify how services of creation, evaluation, dissemination, valuation and maintenance of items of knowledge can be performed on an KMS specific to software testing.

6.3 Main Limitations and Difficulties

Some of the main limitations and difficulties encountered in this work are listed below.

(a) The framework could not be applied in software testing organizations during the project development;

(b) Although two real projects have made their data available to be used, a significant interpretation of not only the data but also the test processes followed by each project was necessary. Besides, it was necessary to seek help of the project leaders and deal with their time limitations.

(c) With regard to the development of Testing KM Portal, all procedures related to the load of knowledge items available in ICAMMH project from MantisBT and TestLink tool, constituted in some difficulties, since there is no reference documentation of these tools test that show an expressive model of their databases.

6.4 Future Work

This work has opened up new opportunities for research field to support the implementation of KM in software testing. However, there are still extensions to be made. Considering the current stage of the work presented here, some of the future directions are highlighted below.

(a) Broaden the coverage of ROoST and evaluate it more accurately;

(b) Mapping the entire ROoST for an Ontology Web Language (OWL);

(c) Apply the framework in ongoing real scenarios to evaluate the results and use them to improve the T-KM framework;

(d) Semantic integration of several testing tools with the Testing KM Portal using ROoST;

(e) Similarly to Yellow Pages, systems for discussion forums, mined items and others should be integrated into the Portal;

(f) Investigate other data mining methods and algorithms to be applied in the knowledge repository of the Portal and to direct more searches in this area;

(g) Improve the Testing KM Portal with respect to non-functional requirements, such as usability, performance, availability;

(h) Conduct experiments.

6.5 Final remarks about this thesis

This section presents some considerations. Several problems were identified with respect to knowledge in software testing within organizations that develop software. Only a very few solutions were found, in the literature, to deal with such issues. These issues can be, in fact, solved at least to a certain extent by employing T-KM framework showed in this thesis. As this is an academic work, many other possibilities could be explored in order to improve the software testing processes within commercial, industrial and academic sectors that have to deal with software development that have to necessarily perform rigorous software testing in order to improve the product quality.

REFERENCES

ABDULLAH, R.; ERI, Z. D.; TALIB, A. M. A model of knowledge management system in managing knowledge of software testing environment. In: MALAYSIAN CONFERENCE IN SOFTWARE ENGINEERING (MYSEC), Johor Bahru, Malaysia. **Proceedings...** Johor Bahru: IEEE, 2011. p. 229–233. 3, 4

ABECKER, A.; BERNARDI, A.; HINKELMAN, K. Toward a technology for organizational memories. **IEEE Intelligent Systems**, German Research Center for Artificial Intelligence, v. 13, No. 3, p. 40–48, 1998. 20

ABECKER, A.; ELST, L. van. Ontologies for knowledge management. **International Handbooks on Information Systems, Springer-Verlag**, p. 435–454, 2004. 48, 52

ABRAN, A.; BOURQUE, P.; DUPUIS JAMES, R.; MOORE, W. **Guide to the software engineering body of knowledge - SWEBOK**. IEEE - 2004 Version, 2004. Available from: <http://www.computer.org/portal/web/swebok/home>. Access in: November, 2013. 2, 12, 13, 14, 69, 80, 83

AHMAD, M. N.; ZAKARIA, N. H.; SEDERA, D. Ontology-based knowledge management for enterprise systems. **International Journal of Enterprise Information Systems**, v. 7, p. 64–90, 2011. 5, 18, 144

ANANDARAJ, A.; KALAIVANI, P.; RAMESHKUMAR, V. Development of ontology-based intelligent system for software testing. **International Journal of Communication, Computation and Innovation**, 2011. 66, 67, 69, 70, 72, 74

ANDRADE, J.; ARES, J.; MARTINEZ, M.-A.; PAZOS, J.; RODRIGUEZ, S.; ROMERA, J.; SUAREZ, S. An architectural model for software testing lesson learned systems. **Information and Software Technology**, University of A Coruña, UDIMA-Madrid Open University, Technical University of Madrid. Spain, v. 55, Nº 1, p. 18–34, 2013. 2, 3, 4, 40, 46

ANTUNES, B.; SECO, N.; GOMES, P. Using ontologies for software development knowledge reuse. In: ARTIFICIAL INTELLIGENCE PORTUGUESE CONFERENCE ON PROGRESS IN ARTIFICIAL INTELLIGENCE, Heidelberg, Germany. **Proceedings...** Heidelberg: Springer, 2007. p. 357–368. 52

ARNICANS, G.; ROMANS, D.; STRAUJUMS, U. Semi-automatic generation of a software testing lightweight ontology from a glossary based on the ONTO6

Methodology. **Frontiers in Artificial Intelligence and Applications**, v. 249, p. 263–276, 2013. 65, 67, 69, 70, 72, 73, 75

BAI, X.; LEE, S.; TSAI, W.; Y., C. Ontology-based test modeling and partition testing of web services. In: INTERNATIONAL CONFERENCE ON WEB SERVICES, Beijing, China. **Proceedings...** Beijing: IEEE, 2008. p. 465–472. 65, 67, 69, 70, 72, 74

BARBOSA, E. F.; NAKAGAWA, E. Y.; MALDONADO, J. C. Towards the establishment of an ontology of software testing. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, San Francisco, CA. **Proceedings...** San Francisco: DBLP, 2006. p. 522–525. 65, 67, 68

BARBOSA, E. F.; NAKAGAWA, E. Y.; RIEKSTIN, A. C.; MALDONADO, J. C. Ontology-based development of testing related tools. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, San Francisco, CA. **Proceedings...** San Francisco: CEUR Workshop Proceedings, 2008. 66, 67, 68

BARCELOS, P. P. F.; SANTOS, V. A.; SILVA, F. B.; MONTEIRO, M. E.; GARCIA, A. S. An automated transformation from OntoUML to OWL and SWRL. In: Seminário de Pesquisa em Ontologias do Brasil (ONTOBRAS), 2013, Belo Horizonte, MG. **Proceedings...** Belo Horizonte: CEUR Workshop Proceedings, 2013. v. 1041, p. 130–141. 95

BASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. **Base de conhecimento em testes de software**. 2. ed. São Paulo: Martins Editora Livraria, 2007. 12, 14, 15

BELLATRECHE, L.; DUNG, N. X.; PIERRA, G. Contribution of ontologybased data modeling to automatic integration of electronic catalogues within engineering databases. **Computers in Industry**, v. 57, p. 711–724, 2006. 57

BENJAMINS, R.; FENSEL, D.; GOMEZ-PEREZ, A. Knowledge management through ontologies. In: THE SECOND INTERNATIONAL CONFERENCE ON PRACTICAL ASPECTS OF KNOWLEDGE MANAGEMENT, Basel, Switzerland. **Proceedings...** Basel: IEEE, 1998. p. 29–30. 2, 5, 18, 22, 52, 143, 144

BLACK, R.; MITCHELL, J. L. **Advanced software testing**: guide to the ISTQB advanced certification as an advanced technical test analyst. 3. ed. USA: Rocky Nook, 2008. 12, 80, 83

BRAGA, G.; ROMANO, B. L.; CAMPOS, H. F.; VIEIRA, R.; CUNHA, A. M.; DIAS, L. A. V. Integrating amazonic heterogeneous hydrometeorological databases. In: INFORMATION TECHNOLOGY: NEW GENERATIONS, 2009. SIXTH INTERNATIONAL CONFERENCE ON, 2009, Las Vegas, NV. **Proceedings...** Las Vegas: IEEE, 2009. p. 119–124. 94

BRINGUENTE, A. C. O.; FALBO, R. A.; GUIZZARDI, G. Using a foundational ontology for reengineering a software process ontology. **Journal of Information and Data Management**, v. 2, p. 511–526, 2011. 60, 165

BUKOWITZ, W.; WILLIAMS, R. L. **The knowledge management fieldbook**. Great Britain: Financial Times Prentice Hall, 1999. ISBN 9780273638827. 17

CAI, L.; TONG, H.; LIU, Z.; ZHANG, J. Test case reuse based on ontology. In: IN PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING, Shanghai, China. **Proceedings...** Shanghai: IEEE, 2009. p. 103 – 108. 66, 67, 69, 70, 72, 74, 75

COELHO, A. G. N. **Uma infraestrutura de gerência de conhecimento em organizações de software aplicada à gestão de riscos**. Master Thesis (Master in Computing) — Universidade Federal do Espírito Santo (UFES), Vitória, ES, Brazil, 2010. 18, 19, 20, 105, 114, 116, 125, 130

D'AQUIN, M.; GANGEMI, A. Is there beauty in ontologies? **Applied Ontology**, v. 6, n. 3, p. 165–175, 2011. 6, 51, 71, 98, 99, 143, 144

DAVENPORT, T. H.; PRUSAK, L. **Working knowledge**: how organizations manage what they know. 2. ed. Boston, USA: Harward Business School Press„ 2000. 15, 17, 18, 113

DAVIES, J.; FENSEL, D.; HARLEMEN, F. V. **Towards the semantic web**: ontology-driven knowledge management. New York, USA: John Wiley & Sons, 2003. 48, 52

DESAI, A. Knowledge management and software testing. In: INTERNATIONAL CONFERENCE AND WORKSHOP ON EMERGING TRENDS IN TECHNOLOGY - TCET, Mumbai, Maharashtra, India. **Proceedings...** Mumbai: IEEE, 2011. p. 767–770. 3, 4, 47

DING, Y.; FENSEL, D. Ontology library systems: the key to successful ontology reuse. In: THE FIRST SEMANTIC WEB WORKING SYMPOSIUM, California, USA. **Proceedings...** California: DBPL, 2001. p. 93–112. 55

DYBA, T.; DINGSOYR, T.; HANSSEN, G. Applying systematic reviews to diverse study types: an experience report. In: INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT (ESEM'07), 1., Madrid, Spain. **Proceedings...** Madrid: IEEE, 2007. p. 225–234. 27, 64

EVERETT, G. D.; RAYMOND, M. J. **Software testing across the entire software development life cycle**. 3. ed. Canada: Published by John Wiley & Sons, Inc, 2007. 15

EVERMANN, J. Towards a cognitive foundation for knowledge representation. **Information Systems Journal**, v. 15, No. 2, p. 147–178, 2005. 59

FALBO, R.; NATALI, A.; MIAN, P.; BERTOLLO, G.; RUY, F. ODE: ontology-based software development environment. In: CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN, La Plata, Argentina. **Proceedings...** La Plata: DBLP, 2003. p. 1124–1135. 105

FALBO, R. A. Experiences in using a method for building domain ontologies. In: INTERNATIONAL WORKSHOP ON ONTOLOGY IN ACTION. INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING, Banff, Canada. **Proceedings...** Banff: DBLP, 2004a. p. 474–477. 79, 99

FALBO, R. A.; ARANTES, D. O.; NATALI, A. C. C. Integrating knowledge management and groupware in a software development environment. In: INTERNATIONAL CONFERENCE ON PRACTICAL ASPECTS OF KNOWLEDGE MANAGEMENT, Vienna, Austria. **Proceedings...** Vienna: Springer, 2004d. p. 94–105. 113

FALBO, R. A.; BARCELLOS, M.; NARDI, J.; GUIZZARDI, G. Organizing ontology design patterns as ontology pattern languages. In: EXTENDED SEMANTIC WEB CONFERENCE, Montpellier, France. **Proceedings...** Montpellier: Springer, 2013. 6, 59, 60, 62, 80, 81, 165, 166, 167, 168, 169, 170

FALBO, R. A.; BERTOLLO, G. A software process ontology as a common vocabulary about software processes. **International Journal of Business Process Integration and Management**, v. 4, p. 239–250, 2009. 165

FALBO, R. A.; MENEZES, A. R. A systematic approach for building ontologies. In: IBERO-AMERICAN CONFERENCE ON PROGRESS IN ARTIFICIAL INTELLIGENCE (IBERAMIA'98), Lisbon, Portugal. **Proceedings...** Lisbon: Springer, 1998b. p. 349–360. 57

FALBO, R. A.; RUY, F. B.; GUIZZARDI, G.; BARCELLOS, M. P.; ALMEIDA, J. P. A. Towards an enterprise ontology pattern language. In: SYMPOSIUM ON APPLIED COMPUTING, Gyeongju, Korea. **Proceedings...** Gyeongju: ACM, 2014. 6, 61, 63, 80, 90, 171, 172, 173

FAYYAD, U.; GREGORY, P.; P.SMYTH, P. Knowledge discovery and data mining: towards a unifying framework. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, Portland, Oregon. **Proceedings...** Portland: IEEE, 1996a. p. 82–88. 113

_____. From data mining to knowledge discovery in databases. **American Association for Artificial Intelligence**, p. 37–54, 1996b. 20, 21

FENSEL, D. **Ontologies**: silver bullet for knowledge management and electronic commerce. 2. ed. New York: Springer, 2003. 22

FENSEL, D.; HARMELEN, F. v.; KLEIN, M.; AKKERMANS, H. On-To-Knowledge: ontology-based tools for knowledge management. In: EBUSINESS AND EWORK 2000 CONFERENCE, Madrid, Spain. **Proceedings...** Madrid, 2000. p. 82–88. 57

FISCHER, G.; OSTWALD, J. Knowledge management: problems, promises, realities, and challenges. **IEEE Intelligent Systems**, v. 16, p. 60–72, 2001. 113

GÓMEZ-PÉREZ, A.; FERNÁNDEZ, M.; VICENTE, A. Towards a method to conceptualize domain ontologies. In: WORKSHOP ON ONTOLOGICAL ENGINEERING, 1996, Budapest, Hungary. **Proceedings...** Budapest: IEEE, 1996. 57

GODBOLE, N. S. **Software quality assurance**: principles and practice. Oxford, UK: Alpha Science International, 2006. 1

GRENON, P.; SMITH, B.; GOLDENBERG, L. Biodynamic ontology: applying BFO in the biomedical domain. **Stud. Health Technol. Inform.**, p. 20–38, 2004. 56

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. **Proceedings...** Padova: ACM, 1993. 2, 55

GRUNINGER, M.; FOX, M. Methodology for the design and evaluation of ontologies. In: WORKSHOP ON BASIC ONTOLOGICAL ISSUES IN KNOWLEDGE SHARING, Montreal, Canada. **Proceedings...** Montreal, 1995. 57

GUARINO, N. Formal ontology and information systems. In: INTERNATIONAL CONFERENCE IN FORMAL ONTOLOGY AND INFORMATION SYSTEMS, Trento, Italy. **Proceedings...** Trento: ACM, 1998. p. 3 – 15. 50, 51, 52, 55, 56

_____. The ontological level: revisiting 30 years of knowledge representation. **Conceptual modelling foundations and applications**, p. 52–67, 1999. 58

GUIZZARDI, G. **Ontological foundations for structural conceptual models**. The Netherlands: Universal Press, ISBN 90-75176-81-3, 2005. 7, 56, 59, 80, 86, 165, 167

_____. On ontology, ontologies, conceptualizations, modeling languages, and (meta)models. In: FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS, DATABASES AND INFORMATION SYSTEMS IV, Amsterdã, The Netherlands. **Proceedings...** Amsterdã: ACM, 2007. p. 18–39. 6, 51, 70

GUIZZARDI, G.; FALBO, R.; GUIZZARDI, R. Grounding software domain ontologies in the unified foundational ontology (UFO): the case of the ODE software process ontology. In: IBEROAMERICAN WORKSHOP ON REQUIREMENTS ENGINEERING AND SOFTWARE ENVIRONMENTS, Recife, PE. **Proceedings...** Recife: DBLP, 2008. p. 127–140. 7, 75, 80, 165

GUIZZARDI, G.; GERD, W.; HERRE, H. On the foundations of UML as an ontology representation language. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE ENGINEERING AND KNOWLEDGE MANAGEMENT, Whittlebury Hall, UK. **Proceedings...** Whittlebury Hall: Springer, 2004. p. 47–62. 59

GUO, S.; ZHANG, J.; TONG, W.; LIU, Z. An application of ontology to test case reuse. In: **Proceedings...** Jilin: IEEE, 2011. p. 19–22. 65, 67, 70, 72, 73

HEIJST, G.; SCHREIBER, A. T.; WIELINGA, B. J. Roles are not classes: a reply to nicola guarino. **International Journal of Human-Computer Studies**, v. 46, p. 311–318, 1997. 55

HENDRIKS, P. H. J.; VRIENS, D. J. Knowledge-based systems and knowledge management: friends or foes? **Information & Management**, v. 35, p. 113–125, 1999. 18

HONG, Z. A framework for service-oriented testing of web services. In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE. DESIGN AND ASSESSMENT OF TRUSTWORTHY SOFTWARE-BASED SYSTEMS, Chicago, USA. **Proceedings...** Chicago: IEEE, 2006. p. 145 – 150. 65, 67

HUO, Q.; ZHU, H.; GREENWOOD, S. A multi-agent software environment for testing web-based applications. In: INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE. DESIGN AND ASSESSMENT OF TRUSTWORTHY SOFTWARE-BASED SYSTEMS, Dallas, TX, USA. **Proceedings...** Dallas: IEEE, 2003. p. 210–215. 65, 67

IEEE. THE INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS (IEEE): standard glossary of software engineering terminology. **IEEE Standard 610.12-1990**, New York, NY, USA, 1990. 1, 11, 80, 87

_____. THE INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS (IEEE): Standard for software test documentation. **IEEE Standard 829-1998**, New York, NY, USA, 1998. 14, 80

ISO/IEC. THE INSTITUTE OF ELECTRIC AND ELECTRONIC ENGINEERS (IEEE): systems and software engineering - software life cycle processes. **ISO/IEC Std 12207-2008**, 2008. 69

JANJIC, W.; ATKINSON, C. Utilizing software reuse experience for automated test recommendation. In: INTERNATIONAL WORKSHOP ON AUTOMATION OF SOFTWARE TEST, San Francisco, USA. **Proceedings...** San Francisco: IEEE, 2013. p. 100–106. 4, 38, 47

JURISTO, N.; FERNDANDEZ, M.; GOMEZ-PEREZ, A. Methontology: from ontological art towards ontological engineering. In: INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE CONFERENCE, Stanford, USA. **Proceedings...** Stanford: Springer, 1997. p. 33–40. 69

KERKHOF, C.; ENDE, J.; BOGENRIEDER, I. Knowledge management in the professional organization: a model with application to CMG software testing. **Knowledge and Process Management**, v. 10, No. 2, p. 77–84, 2003. 4

KIM, H. M. Developing ontologies to enable knowledge management: integrating business process and data driven approaches. In: WORKSHOP ON BRINGING KNOWLEDGE TO BUSINESS PROCESSES, Stanford University. **Proceedings...** Stanford: Spring, 2000. 2, 48, 52, 143

KITCHENHAM, B. A. **Guidelines for performing systematic literature reviews in software engineering**. Keele University, 2007. Technical Report EBSE-2007-01. Available from: <http://urlib.net/8JMKD3MGP7W/38CDFNB>. Access in: September, 2013. 22, 24

KITCHENHAM, B. A.; BUDGEN, D.; BRERETON, O. Using mapping studies as the basis for further research: a participant-observer case study. **Journal of Information and Software Technology**, p. 638–651, 2011. 5, 22

KOSCIANSKI, A.; SANTOS, M. **Qualidade de software**: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2. ed. São Paulo: Novatec Editora, 2007. 1

LAMAS, E.; SOUZA, E. F.; NASCIMENTO, M. R.; DIAS, L. A. V.; SILVEIRA, F. F. Organizational testing management maturity model for a software product line. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY, IEEE COMPUTER SOCIETY, Las Vegas, NV. **Proceedings...** Las Vegas: IEEE, 2010b. 3

LAMAS, E. A. **Uma estrutura de maturidade operacional para gestão de teste de software aplicada a um projeto de monitoramento hidrológico**. Master Thesis (Master in Computer Science) — Instituto Técnologico de Aeronáutica (ITA), São José dos Campos, SP, Brazil, 2010a. 3

LEVESON, N. G.; TURNER, C. S. An investigation of the therac-25 accidents. **Computer**, v. 26, p. 18–41, 1993. 1

LI, X.; PARSONS, J. Ontological semantics for the use of UML in conceptual modeling. In: TUTORIALS, POSTERS, PANELS AND INDUSTRIAL CONTRIBUTIONS AT THE INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, 2007, Darlinghurst, Australia. **Proceedings...** Darlinghurst: ACM. p. 179–184. 59

LI, X.; ZHANG, W. Ontology-based testing platform for reusing. In: INTERNATIONAL CONFERENCE ON INTERNET PLATFORM FOR REUSING, Henan, China. **Proceedings...** Henan: IEEE, 2012. p. 86–89. 3, 4, 43, 48, 49, 50, 65, 67, 70, 73

LIAO, S. Knowledge management technologies and applications-literature. **Expert Systems with Applications**, v. 25, p. 155–164, 2003. 17

LIU, Y.; WU, J.; LIU, X.; GU, G. Investigation of knowledge management methods in software testing process. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY AND COMPUTER SCIENCE, Kiev, Ukraine. **Proceedings...** Kiev: IEEE, 2009. p. 90–94. 3, 47, 49

MAEDCHE, A.; MOTIK, B.; STOJANOVIC, L.; STUDER, R.; VOLZ, R. Ontologies for enterprise knowledge management. **IEEE Intelligent Systems**, p. 26–33, 2003. 18

MAEDCHE, A.; VOLZ, R. The text-to-onto ontology extraction and maintenance environment to appear. In: WORKSHOP ON INTEGRATIN DATA MINING AND KNOWLEDGE MANAGEMENT, San Jose, California. **Proceedings...** San Jose, 2001. 5, 144

MANTIS. 2013. Available from: <http://www.mantisbt.org/>. Access in: November, 2013. 118

MATHUR, A. P. **Foundations of software testing**. 5. ed. India: Dorling Kindersley (India), Pearson Education in South Asia, 2012. 1, 12, 13, 14, 80, 83, 89

MONTONI, M. A. **Aquisição de conhecimento**: uma aplicação no processo de desenvolvimento de software. Master Thesis (Master in Computer Science) — Universidade Federal do Rio de Janeiro - COPPE/UFRJ, Rio de Janeiro, RJ, Brazil, 2003. 18, 19, 130

MYERS, G. J. **The art of software testing**. 2. ed. Canada: John Wiley and Sons, 2004. 12, 13, 80

NAKAGAWA, E. Y.; BARBOSA, E. F.; MALDONADO, J. C. Exploring ontologies to support the establishment of reference architecture: an example on software testing. In: WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE (WICSA)/EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE (ECSA), Cambridge, UK. **Proceedings...** Cambridge: IEEE, 2009. p. 249–252. 65, 67

NASA. **NASA jet propulsion laboratory**: mars climate orbiter mission. Cape Canaveral Air Force Station, Florida, 2014. Available from: <http://www.jpl.nasa.gov/missions/mars-climate-orbiter/>. Access in: January, 2014. 1

NASSER, V. H.; DU, W.; MACISAAC, D. Knowledge-based software test generation. In: INTERNATIONAL CONFERENCE ON SOFTWARE

ENGINEERING AND KNOWLEDGE ENGINEERING, Boston, Massachusetts. **Proceedings...** Boston: DBLP, 2009. p. 312–317. 65, 67, 70, 72, 74

NATALI, A.; ROCHA, A.; TRAVASSOS, G.; MIAN, P. Integrating verification and validation techniques knowledge into software engineering environments. In: JORNADAS IBEOAMERICANAS DE INGENIERíA DEL SOFTWARE E INGENIERíA DEL CONOCIMIENTO, Madrid, Spain. **Proceedings...** Madrid, 2004. p. 419–430. 3

NATALI, A. C. C. **Uma Infraestrutura para gerência de conhecimento em um ambiente de desenvolvimento de software.** Master Thesis (Master in Computer Science) — Universidade Federal do Espirito Santo (UFES), Vitória, ES, Brazil, 2003. 18, 20, 113

NOGESTE, K.; WALKER, D. H. T. Using knowledge management to revise software-testing processes. **Journal of Workplace Learning**, v. 18, p. 6–27, 2003. 3, 38

NONAKA, I.; KROGH, G. Tacit knowledge and knowledge conversion: controversy and advancement in organizational knowledge creation theory. **Organization Science**, v. 30, p. 635–652, 2009. 16, 17, 113

NONAKA, I.; TAKEUCHI, H. **The knowledge-creating company**: how japanese companies create the dynamics of innovation. 1. ed. USA: Oxford University Press, Oxford, 1997. 15, 16, 35, 42, 47

NOY, N.; MCGUINNESS, D. **Ontology development 101 a guide to creating your first ontology**. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI, 2001. Available from: <http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.136.5085&rep=rep1&type=pdf>. Access in: November, 2013. 69

NUNAMAKER, J.; ROMANO, N.; BRIGGS, R. Framework for collaboration and knowledge management. In: ANNUAL HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, Washington, DC, USA. **Proceedings...** Washington: IEEE, 2001. v. 1. 18

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. 55

O'LEARY, D.; STUDER, R. Knowledge management: an interdisciplinary approach. **IEEE Intelligent Systems**, v. 16, No. 1, 2001. 17, 22

O'LEARY, D. E. Enterprise knowledge management. **IEEE Computer Magazine**, p. 54–61, 1998a. 3, 17, 18, 19, 21, 22, 113

OWL. **OWL web ontology language**. Word Wide Web Consortium (W3C), 2003. Available from: <http://www.w3.org/TR/2003/WD-owl-ref-20030331/>. Access in: Sep. 2013. 59

PEASE, A.; NILES, I.; LI, J. The suggested upper merged ontology: a large ontology for the semantic web and its applications. In: WORKSHOP ON ONTOLOGIES AND THE SEMANTIC WEB, Edmonton, Canada. **Proceedings...** Edmonton: IEEE, 2002. 56

PECHEUR, C. Verification and validation of autonomy software at NASA. **NASA Ames Research**, Moffett Field, USA, 2000. 1

PERRY, W. E. **Effective methods for software testing**. 3. ed. Canada: Wiley Publishing, Inc., 2006. 15

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, Bari, Italy. **Proceedings...** Bari: IEEE, 2008. p. 68–77. 22, 25, 32, 34

PRESSMAN, R. S. **Software engineering**: a practitioner's approach. 6. ed. New York: McGraw-Hill series in computer science, 2006. 12, 80

PRESUTTI, V.; DAGA, E.; GANGEMI, A.; BLOMQVIST, E. Extreme design with content ontology design patterns. In: WORKSHOP ON ONTOLOGY PATTERNS, Washington, USA. **Proceedings...** Washington: CEUR Workshop Proceedings, 2009. 60

PROBST, G.; RAUB, S.; ROMHARDT, K. **Managing knowledge**: building blocks for success. Chichester, England: John Wiley & Sons, 2000. 18

RDF. **Resource description framework (RDF)**. Word Wide Web Consortium (W3C), 2004. Available from: <http://www.w3.org/RDF/>. Access in: September, 2013. 59

RIOS, J. A. Ontologias: alternativa para a representação do conhecimento explícito organizacional. In: ENCONTRO NACIONAL DE CIÊNCIA DE INFORMAÇÃO, Salvador, Bahia. **Proceedings...** Salvador: DBPL, 2005. 55

RUBENSTEIN-MONTANO, B.; LIEBOWITZ, J.; BUCHWALTER, J.; MCCAW, D.; NEWMAN, B.; K., R. Smartvision:a knowledge-management methodology. **Journal of Knowledge Management**, v. 5, n. 2, p. 300 – 310, 2000. 77

RUGGLES, R. The state of the notion: knowledge management in practice. **California Management Review**, v. 40, n. 3, p. 80–89, 1998. 19

RYU, H.; RYU, D.; BAIK, J. A strategic test process improvement approach using an ontological description for MND-TMM. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCE, Portland, OR. **Proceedings...** Portland: IEEE, 2011. p. 561–566. 65, 67, 69, 70, 72, 74

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

SANTIAGO JÚNIOR, V. A. and VIJAYKUMAR, N. L. and SOUZA, E. F. and GUIMARÃES, D. and COSTA, R. C. GTSC: automated model-based test case generation from statecharts and finite state machines. In: Tools Session of the III Congresso Brasileiro de Software: Teoria e Prática (CBSoft), Natal, RN. **Proceedings...** Natal, 2012. p. 25–30. 3

SAPNA, P. G.; MOHANTY, H. An ontology based approach for test scenario management. **Communications in Computer and Information Science**, v. 141, p. 91–100, 2011. 66, 67, 69, 70, 72, 74

SOUZA, E. F.; EVARISTO, L.; VIJAYKUMAR, N. L. Ontology in software testing: a systematic literature review. In: Seminário de Pesquisa em Ontologias do Brasil (ONTOBRAS), Belo Horizonte, MG. **Proceedings...** Belo Horizonte: CEUR Workshop Proceedings, 2013c. v. 1041, p. 71–82. 6, 63

SOUZA, E. F.; FALBO, R. A.; VIJAYKUMAR, N. L. Knowledge management applied to software testing: a systematic mapping. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE

ENGINEERING (SEKE 2013), Boston, USA. **Proceedings...** Boston: IEEE, 2013a. p. 562–567. 4, 5, 7, 21

_____. Using ontology patterns for building a reference sofware testing ontology. In: INTERNATIONAL WORKSHOP ON VOCABULARIES, ONTOLOGIES AND RULES FOR THE ENTERPRISE AND BEYOND (VORTE 2013), Vancouver, BC. **Proceedings...** Vancouver: IEEE, 2013b. p. 21–30. 6, 76, 143

SOUZA, E. F.; SANTOS, R. **Uso de mineração de dados para análise do processo de teste**. Technical Report, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, SP, Brazil, 2010. Available from: <http://urlib.net/8JMKD3MGP7W/38CDFNB>. Access in: November, 2013. 112, 120

SPECIMILLE, M. S.; FALBO, R. A.; SOUZA, E. F. TKMP: um portal para gerencia de conhecimento em teste de software. Work to be submitted in Computer Science for obtaining Bachelor Degree. Universidade Federal do Espírito Santo (UFES), Vitória, ES, Brazil. 2014. 125

STAAB, S.; MAEDCHE, A. Ontology engineering beyond the modeling of concepts and relations. In: EUROPEAN CONFERENCE ON ARTIFICIAL INTELLIGENCE (ECAI). WORKSHOP ON APPLICATIONS OF ONTOLOGIES AND PROBLEM-SOLVING METHODS, Berlin, Germany. **Proceedings...** Berlin, 2000. 52

STAAB, S.; STUDER, R.; SCHURR, H. P.; SURE, Y. Knowledge processes and ontologies. **Intelligent Systems**, v. 16, p. 26–34, 2001. 2, 3, 18, 22, 48, 143

STOREY, J.; BARNETT, E. Knowledge management initiatives: learning from failure. **Journal of Knowledge Management**, v. 4, n. 2, p. 145 – 156, 2000. 77

SWARTOUT, B.; PATIL, R.; KNIGHT, K.; RUSS, T. Toward distributed use of large-scale ontologies. In: KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP, Banff, Canada. **Proceedings...** Banff: Springer, 1996. 57

TESTLINK. 2013. Available from: <http://testlink.org/>. Access in: November, 2013. 118

THRANE, C. **Quantitative models and analysis for reactive systems**. PhD Thesis (Thesis in Computer Science) — Department of Computer Science - Aalborg University, Denmark, 2011. Available from:

<http://people.cs.aau.dk/~crt/Thrane-thesis.pdf>. Access in: October, 2013. 1

USCHOLD, M. Building ontologies: towards a unified methodology. In: ANNUAL CONF. OF THE BRITISH COMPUTER SOCIETY SPECIALIST GROUP ON EXPERT SYSTEMS, Cambridge, UK. **Proceedings...** Cambridge, 1996b. p. 16–18. 55

USCHOLD, M.; GRUNINGER, M.; USCHOLD, M.; GRUNINGER, M. Ontologies: principles, methods and applications. **Knowledge Engineering Review**, v. 11, p. 93–136, 1996a. 55

USCHOLD, M.; JASPER, R. A framework for understanding and classifying ontology applications. In: WORKSHOP ON ONTOLOGIES AND PROBLEM, Stockholm, Sweden. **Proceedings...** Stockholm: CEUR Workshop Proceedings, 1999. p. 1–11. 55

USCHOLD, M.; KING, M. Towards a methodology for building ontologies. In: PRESENTED AT THE WORKSHOP ON BASIC ONTOLOGICAL ISSUES IN KNOWLEDGE SHARING, IJCAI'95, Edinburgh, Scotland. **Proceedings...** Edinburgh, 1995. 69

VALASKI, O.; MALUCELLI, A.; REINEHR, S. Ontologies application in organizational learning: a literature review. **Expert Systems with Applications**, v. 39, p. 7555–7561, 2012. 5, 18, 144

W3C. **World Wide Web Consortium**. 2013. Available from: <www.w3c.br>. Access in: September, 2013. 59

WIERINGA, R.; MAIDEN, N.; MEAD, N.; ROLLAND, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. **Requirements Engineering**, v. 11, p. 102–107, 2006. 25, 34, 39

WINCH, G. Knowledge management. **Manufacturing Engineer**, v. 78, p. 178–180, 1999. 19

WITTEN, I. H.; FRANK, E.; HALL, M. A. **Data mining**: practical machine learning tools and techniques. 2. ed. San Francisco: Morgan Kaufmann, 2005. 120, 121

WONG, K. Y.; ASPINWALL, E. Knowledge management implementation frameworks: a review. **Knowledge and Process Management**, v. 11, n. 2, p. 93–104, 2004. 7, 77

WU, Y. L. J.; XUEMEI, L. G. G. Investigation of knowledge management methods in software testing process. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY AND COMPUTER SCIENCE, Kiev, Ukraine. **Proceedings...** Kiev: IEEE, 2009. p. 90–94. 3, 4

XU-XIANG, L.; WEN-NING, Z. The PDCA-based software testing improvement framework. In: INTERNATIONAL CONFERENCE ON APPERCEIVING COMPUTING AND INTELLIGENCE ANALYSIS, Chengdu, China. **Proceedings...** Chengdu: IEEE, 2010. p. 490–494. 3, 43

YU, L.; SU, S.; ZHAO, J. Performing unit testing based on testing as a service (taas) approach. Shing-Chi Chenug, p. 127–131, 2008. 65, 67, 75

YU, L.; XIANG, H.; SU, Y.; ZHAO, W.; ZHU, J. A framework of testing as a service. IEEE, Wuhan, p. 1–4, 2009. 65, 67, 75

YUFENG, Z.; HONG, Z. Ontology for service oriented testing of web services. In: INTERNATIONAL SYMPOSIUM ON SERVICE-ORIENTED SYSTEM ENGINEERING, Jhongli, Taiwan. **Proceedings...** Jhongli: IEEE, 2008. p. 129 – 134. 65, 67

ZACK, M.; SERINO, M. Knowledge management and collaboration technologies. In: KNOWLEDGE, GROUPWARE AND THE INTERNET, Butterworth, Heinemann. **Proceedings...** Butterworth, 2000. p. 303–315. 17

ZAMBORLINI, V. **Estudo de alternativas de mapeamento de ontologias linguagem OntoUML para OWL**: abordagens para representação de informação temporal. Master Thesis (Master in Computer Science) — Universidade Federal do Espírito Santo (UFES), Vitória, ES, Brazil, 2011. 95, 97

ZHU, H.; HUO, Q. Developing a software testing ontology in UML for a software growth environment of web-based applications. **Software Evolution with UML and XML**, Hongji Yang, p. 263–295, 2005. 65, 67

ZHU, H.; ZHANG, Y. Collaborative testing of web services. **IEEE Transactions on Service Computing**, v. 5, p. 116–130, 2012. 65, 67

ANNEX A - Ontology Pattern Language: SP-OPL and E-OPL

This annex describes some patterns from Software Process Ontology Pattern Language (SP-OPL)and Enterprise Ontology Pattern Language (E-OPL) which were used in the development of ROoST.

Software Process Ontology (SPO) was originally presented in (FALBO; BERTOLLO, 2009), and afterwards, in (BRINGUENTE et al., 2011), partially reengineered at the light of the Unified Foundational Ontology (UFO) (GUIZZARDI, 2005; GUIZZARDI et al., 2008). According to Bringuente et al. (2011), the use of UFO was useful in identifying problems and for driving the ontology reengineering, mainly by describing ontological commitments that were implicit.

In Falbo et al. (2013), SPO is presented as a core ontology on software processes, called SP-OPL. As a core ontology, SP-OPL provides a precise definition of the structural knowledge in the field of software processes that spans across different application domains in this field (FALBO et al., 2013). Moreover, SP-OPL is built grounded on the UFO.

As an ontology pattern language, SP-OPL contains a set of interrelated ontology patterns related to the software process domain, plus a map providing explicit guidance on what problems can arise in this universe of discourse, informing the order to address these problems, and suggesting one or more patterns to solve each specific problem (FALBO et al., 2013).

SP-OPL organizes 30 patterns and has three entry points, i.e. three different ways to enter in the pattern language. The choice of the entry point from which to enter in the SP-OPL depends on the focus of the ontology engineer. When the requirements for the domain ontology being developed include problems related to definition of standard software processes, the entry point is the Standard Process Structure (SPS) pattern, from which other patterns related to the definition of standard software processes can be achieved. The second entry point is the Software Process Planning (SPP) pattern, which deals with how a software process is planned in terms of subprocesses and activities. From this pattern, other patterns related to the definition of a software process for a project and scheduling it can be achieved. Finally, the third entry point in SP-OPL is the PAE pattern, which deals with representing process and activity occurrences. From this pattern, the ontology engineer can achieve others that address problems related to resource participation, procedures adopted, and work product inputs and outputs (FALBO et al., 2013).

For developing ROoST, the third entry point (EP3) of SP-OPL was used. Figure A.1 shows the SP-OPL patterns accessible from this entry point. The PAE pattern represents the occurrences of processes and activities in the context of a project. The HRPA pattern represents the participation of a human resource in an activity occurrence. The RPA pattern represents the participations of resources (hardware and software) in activity occurrences. The WPPA pattern represents the participations of artifacts in activity occurrences. Finally, the PRPA pattern represents the participation (adoption) of procedures in activity occurrences.

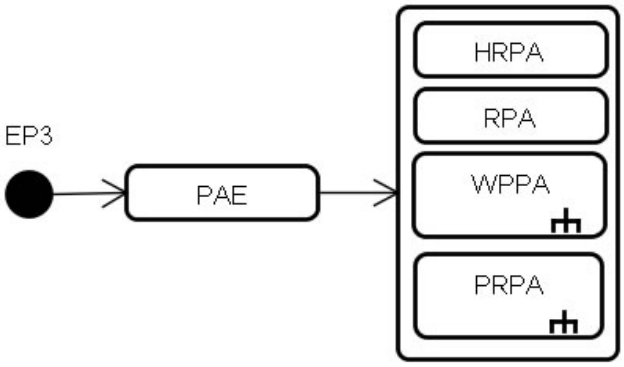

Figure A.1 - SP-OPL patterns accessible from the entry point EP3
SOURCE: (FALBO et al., 2013)

Besides the 30 patterns described in the SP-OPL map, SP-OPL has two supplementary patterns: Work Product Taxonomy (WPT), which describes types of artifacts, and Procedure Taxonomy (PRT), which describes types of procedures. Figures A.2 and A.3 present these patterns.

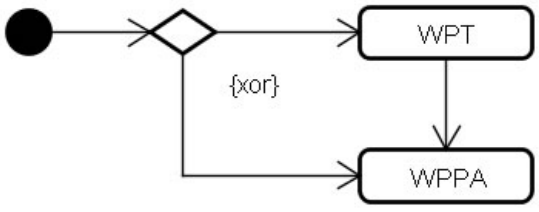

Figure A.2 - Work Product Taxonomy (WPT)
SOURCE: (FALBO et al., 2013)

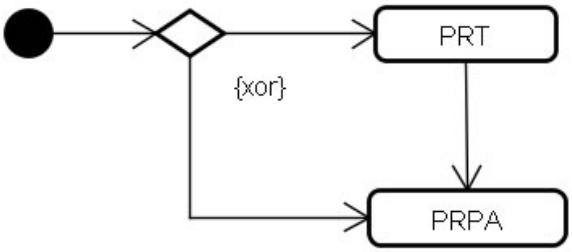

Figure A.3 - Procedure Taxonomy (PRT)
SOURCE: (FALBO et al., 2013)

Patterns in SP-OPL are described using a form, including: name, intent, competency questions that the pattern aims to answer, conceptual model, axiomatization, and foundations (ontological analysis taking ontological distinctions of UFO into account). The conceptual models of the SP-OPL patterns are encoded in OntoUML (GUIZZARDI, 2005), a UML profile that enables modelers to make finer-grained modeling distinctions between different types of classes and relations according to some ontological distinctions put forth by UFO.

Figure A.4 shows the conceptual model of PAE pattern (FALBO et al., 2013). The intent of this pattern is to represent the occurrences of processes and activities in the context of a project, and their mereological structure. The following competency questions are addressed by this pattern: (PAE-CQ1) *What is the project in which context a given process/activity occurrence occurred?*; (PAE-CQ2) *How is a process occurrence structured in terms of sub-processes and activities?*; (PAE-CQ3) *When did a process occurrence start and when did it end?*; (PAE-CQ4) *When did an activity occurrence start and when did it end?*; (PAE-CQ5) *From which activity occurrences does an activity occurrence depend on?*

Process Occurrences and Activity Occurrences are complex events, and the whole-part relations between events are strict partial order. In the software process domain, there are two main kinds of Process Occurrences: General Process Occurrence and Specific Process Occurrence. A general process occurrence is the whole execution of the process defined for a Project. It is composed by specific process occurrences, allowing an organization to decompose a general process into sub-processes. A specific process occurrence, in turn, is decomposed into Activity Occurrences. Activity occurrences can be simple or composite. A composite activity occurrence is a complex event that is composed by other activity occurrences. A simple activity occurrence is not composed by other activity occurrences, but it is still a complex event in

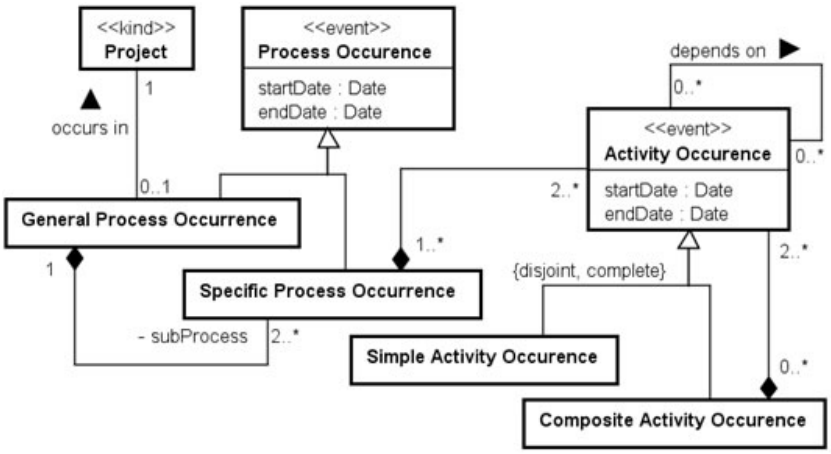

Figure A.4 - The Process and Activity Execution PAE ontology pattern
SOURCE: (FALBO et al., 2013)

UFO, since it is composed by other events representing the participations of human resources, hardware and software resources, artifacts, and procedures in the activity occurrence (FALBO et al., 2013).

Figure A.5 presents the conceptual model of the WPPA pattern (FALBO et al., 2013). The intent of this pattern is to represent the participation of artifacts (as input or output) in an activity occurrence. The following competency questions are addressed by this pattern: (WPPA-CQ1) *Which artifacts are produced in (are an output of) an activity occurrence?*; (WPPA-CQ2) *Which artifacts are used in (are an input to) an activity occurrence?*; (WPPA-CQ3) *When did an artifact participation in an activity occurrence start and when did it end?*.

An Activity Occurrence can have as its parts Artifact Participations, which are also events. An Artifact Participation is the participation of a single Artifact in an Activity Occurrence. Artifact participations can be of three types: (i) Artifact Creation, meaning that the artifact is created during the activity occurrence, and thus it is an output of this activity occurrence; (ii) Artifact Usage, meaning that the artifact is only used during the activity occurrence, and thus it is only an input for the activity occurrence; and (iii) Artifact Change, meaning that the artifact is changed during the activity occurrence, and thus it is both input and output for the activity occurrence. It is worthwhile to point out that both produces and uses relations between Activity Occurrence and Artifact are derived from the participations of the artifacts in the activity occurrence. Thus they are represented preceded by a bar (/).

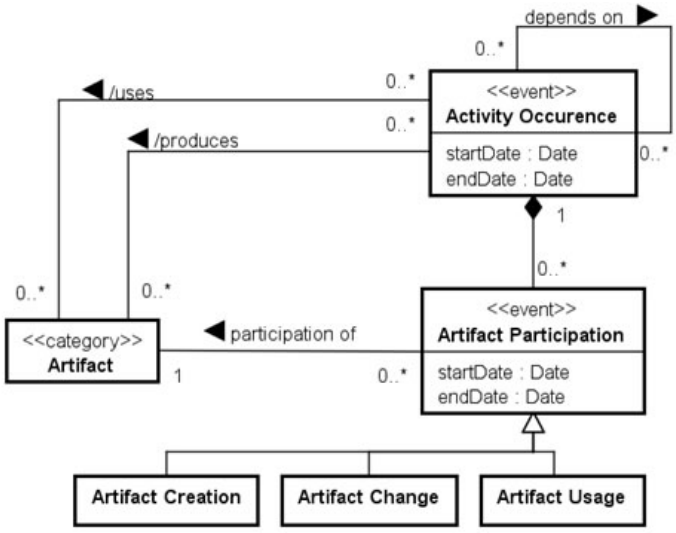

Figure A.5 - The Work Product Participation (WPPA)ontology pattern.
SOURCE: (FALBO et al., 2013)

Figure A.6 presents the conceptual model of the PRPA ontology pattern. During an activity occurrence, Procedures are adopted, giving rise to Procedure Participations. The competency question addressed by this pattern: (PRPA-CQ1) *Which are the procedures adopted in an activity occurrence?*

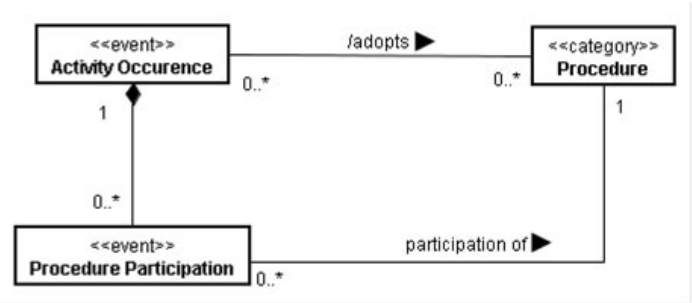

Figure A.6 - Procedure Participation (PRPA) ontology pattern.
SOURCE: (FALBO et al., 2013)

Figure A.7 presents the conceptual model of the PRPA ontology pattern. The intent of this pattern is to represent participations of human resources in activity occurrences. The following competency questions are addressed by this pattern: (PRPA-CQ1) *Which human resources participate in an activity occurrence?*; (PRPA-CQ2) *When a human resource participation started and when it ended?*.

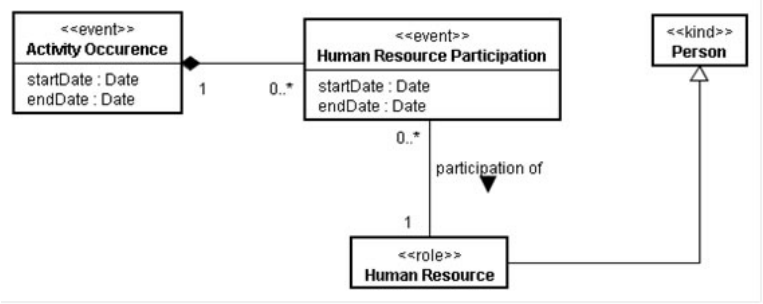

Figure A.7 - Human Resource Participation (HRPA) ontology pattern.
SOURCE: (FALBO et al., 2013)

Figure A.8 presents the conceptual model of the RPA ontology pattern. The intent of this pattern is to represent participations of resources (hardware and software) in activity occurrences. The following competency questions are addressed by this pattern: (RPA-CQ1) *Which hardware resources are used in an activity occurrence?*; (RPA-CQ2) *When a hardware resource participation started and when it ended?*; (RPA-CQ3) *Which software resources are used in an activity occurrence?*; (RPA-CQ4) *When a software resource participation started and when it ended?*.

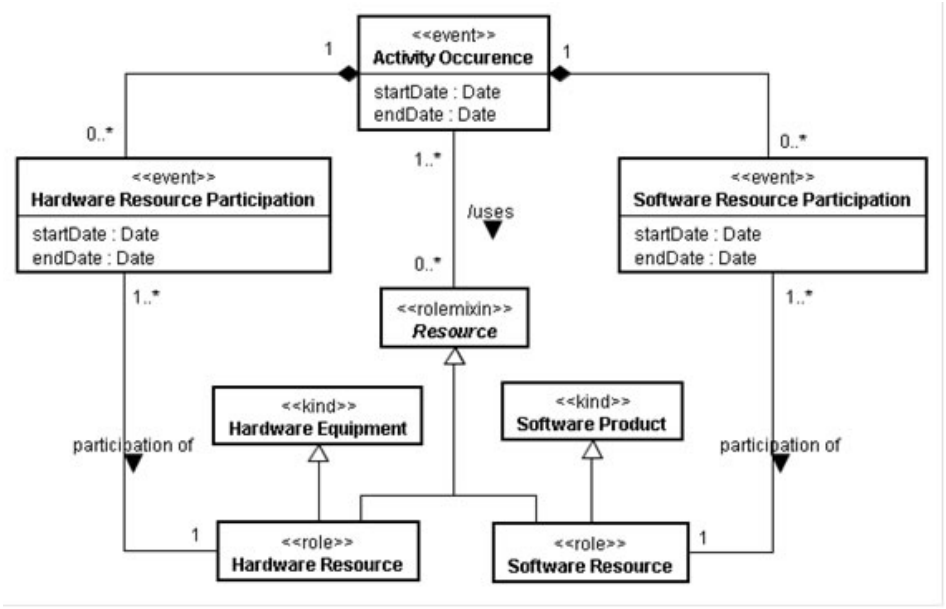

Figure A.8 - Resource Participation (RPA) ontology pattern.
SOURCE: (FALBO et al., 2013)

In order to address aspects common to several enterprises, the E-OPL presents some aspects such as Organization Arrangement, Definition Team, Institutional Roles, Institutional Goals, and Human Resource Management. Each one aspect provides a set of patterns. As Figure A.9 shows, E-OPL has two entry points. *EP1* can be chosen when the requirements for the new enterprise ontology being developed include only problems related to the definition of project teams. Otherwise, the starting point is *EP2* that address problems related to how an organization is structured. Patterns Multi-Organization Arrangement (MOAR), Organizational Team Definition (OTD), Team Roles (TEAR) and Team Allocation (TEAA) were reused in the development of the ROoST.

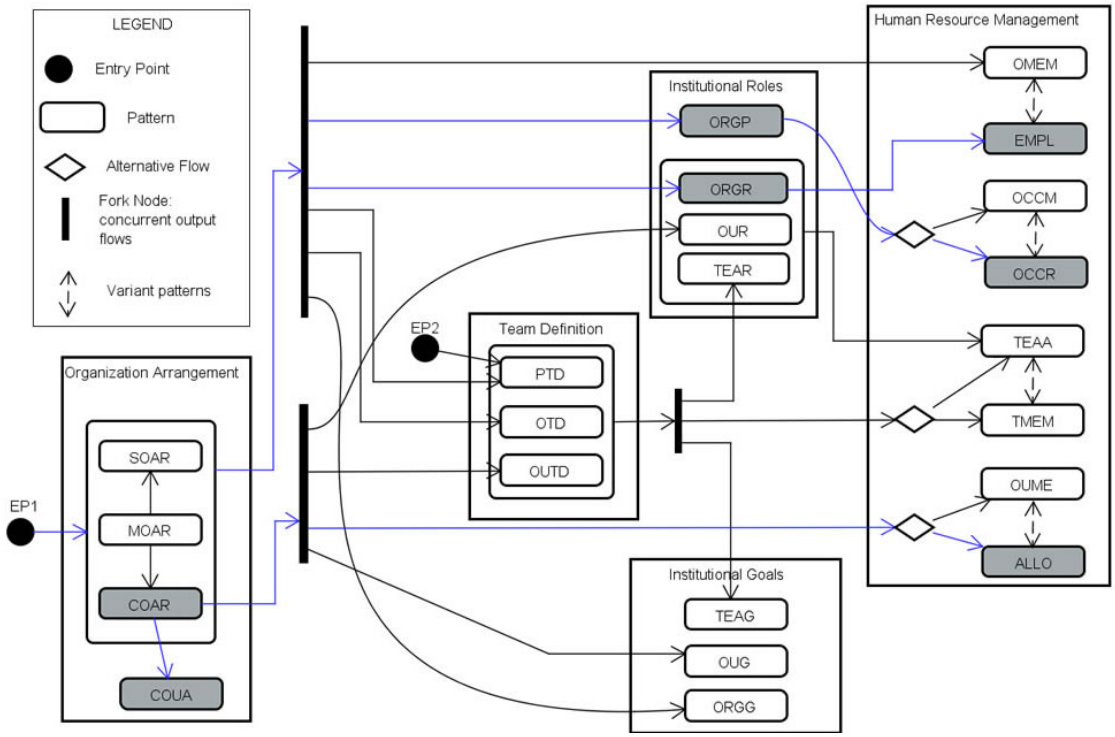

Figure A.9 - Enterprise-Ontology Pattern Language (E-OPL)
SOURCE: (FALBO et al., 2014)

Multi-Organization Arrangement (MOAR) should be selected as the first pattern if the ontology engineer needs to represent organizations that are composed of other organizations. Figure A.10 presents the conceptual model of the MOAR ontology pattern.

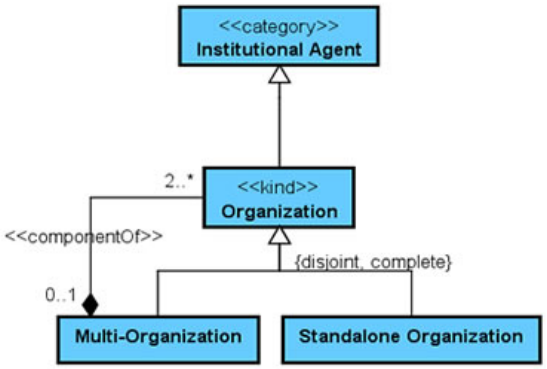

Figure A.10 - Multi-Organization Arrangement (MOAR)
SOURCE: (FALBO et al., 2014)

Concerning team definition, the Organizational Teams (OTD) ontology pattern was reused. Figure A.11 presents the conceptual model of the OTD ontology pattern.

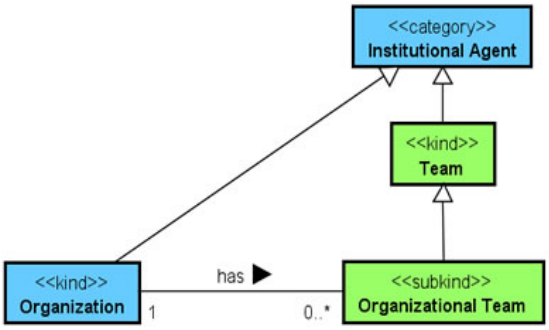

Figure A.11 - Organizational Teams (OTD)
SOURCE: (FALBO et al., 2014)

In order to deal with institutional roles, one of the patterns was reused is the Team Roles TEAR that concern informal roles defined by an organizational unit or a team. Figure A.12 presents the conceptual model of the TEAR ontology pattern.

Figure A.12 - Team Roles (TEAR)
SOURCE: (FALBO et al., 2014)

Regarding team allocation, Team Allocation (TEAA) was reused. This pattern considers the relator *Team Allocation* and three mediation relations between this relator and *Human Resource*, *Team* and the *Human Resource Role* that the human resource plays in that team. Figure A.13 presents the conceptual model of the TEAA ontology pattern.

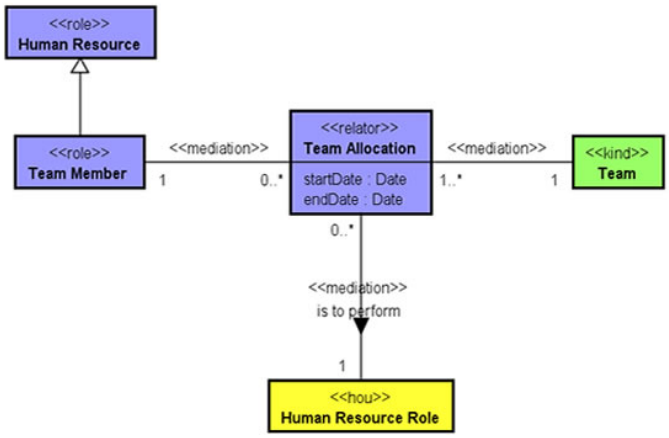

Figure A.13 - Team Allocation (TEAA)
SOURCE: (FALBO et al., 2014)

APPENDIX A - Survey: KM in software testing

This survey aimed to identify a specific scenario the software testing domain to exercise the KM. This research took into account the answers given by experts in the Software Engineering area with experience in Software Testing. A summary of these survey results were presented in section 5.1. This appendix presents the questions, the tables that compiled the answers to each question as well as some results aspects obtained by applying the survey.

General Information: Name, Role, Time Experience.

Question 01. In which activities of a Testing Process, is KM more useful?

Figure A.1 - Question 01. Importance of KM to Software Testing Process Activities

In this question, the testing activities showed in Figure A.1 were considered. Table A.1 shows the percentage of answers per activity. "Test Case Design" (98.84%) and "Planning Test" (96.51%) have the largest representativeness.

Table A.1 - Importance of KM to Software Testing Process Activities

Test Planning				
Very Important	58	Very Important	83	96.51%
Important	25			
Not very Important	2	Not very Important	3	3.49%
Not Important	1			
Test Case Design				

Continues

Table A.1 - Conclusion

Very Important	46	Very Important	85	98.84%
Important	39			
Not very Important	0	Not very Important	1	1.16%
Not Important	1			
Test Code				
Very Important	24	Very Important	59	68.60%
Important	35			
Not very Important	25	Not very Important	27	31.40%
Not Important	2			
Test Execution				
Very Important	29	Very Important	55	63.95%
Important	26			
Not very Important	26	Not very Important	31	36.05%
Not Important	5			
Test Result Analysis				
Very Important	37	Very Important	75	87.21%
Important	38			
Not very Important	9	Not very Important	11	12.79%
Not Important	2			

Question 02. In which activities of Testing Planning, is KM more useful?

( ) Testing Technique Selection
( ) Test Environment Definition (including hardware, software and human resources that should be part of the testing environment)
( ) Test Plan Elaboration
( ) Other

Table A.2 presents the percentage of answers per sub-activity of testing planning. The usefulness of KM for selecting the best testing techniques was recognized by 41% of the participants, since different types of test techniques determine different forms for selecting the test cases that will be used as input to the system under examination.

Table A.2 - Usefulness of KM in sub-activities of Testing Planning

Activity	**Amount**	**Percentage**
Testing Technique Selection	35	41.70%
Test Environment Definition	31	36.05%
Test Plan Elaboration	18	20.93%
Other	2	2.33%

Question 03. A test environment consists of, among others, human resources, hardware and software. About which of these resources are more important to have available knowledge at the moment of defining the test environment?

( ) Human Resource
( ) Software Resource
( ) Hardware Resource
( ) Other

Table A.3 presents the percentage of answers per test environment resources. "Human resource" is considered the most important at the time of setting the test environment, with approximately 44% of answers.

Table A.3 - Test Environment Resources

Resources	**Amount**	**Percentage**
Human Resource	38	44.19%
Software Resource	35	40.70%
Hardware Resource	9	10.47%
Other	4	4.65%

Question 04. In which testing level is KM more useful?

( ) Unit Testing
( ) Integration Testing
( ) System Testing
( ) Other

The experts consider that in the software testing process, KM can be more useful in

the System Testing level (approximately 49%). Table A.4 presents the percentage of answers per test level.

Table A.4 - Importance of KM to Test Levels

Test Level	**Amount**	**Percentage**
Unit Testing	8	9.30%
Integration Testing	31	36.05%
Hardware Resource	42	48.84%
System Testing	5	5.81%

Question 05. What is the type of knowledge you consider to be more important during the software testing process?

( ) Tacit Knowledge
( ) Explicit Knowledge

Table A.5 presents the percentage of answers per type of knowledge. In this question explicit knowledge was considered more important by most experts with approximately 70% of answers.

Table A.5 - Type of knowledge

Type of knowledge	**Amount**	**Percentage**
Tacit Knowledge	26	30.23%
Explicit Knowledge	60	69.8%

Question 06. Tacit knowledge can be made explicit, originating explicit knowledge. Regarding the types of knowledge items listed below, indicate the degree of importance of generating explicit knowledge from tacit knowledge.

In this question, “Individual Experiences” (95.35%) and “Communications between the members of the test team” (approximately 92%) are the types of tacit knowledge with more significant importance to generate explicit knowledge. Figure A.2 shows the knowledge items considered. Table A.6 shows the percentage of answers.

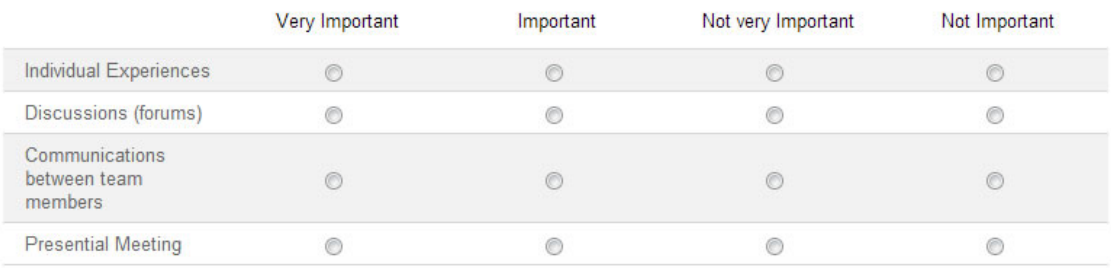

Figure A.2 - Question 06. Making Tacit Knowledge Explicit

Table A.6 - Making Tacit Knowledge Explicit

Individual Experiences				
Very Important	45	Very Important	82	95.3%
Important	37			
Not very Important	3	Not very Important		8.14%
Not Important	1			
Discussions (forums)				
Very Important	23	Important	72	83.72%
Important	32			
Not very Important	13	Not very Important	14	16.28%
Not Important	1			
Communications between team members				
Very Important	54	Very Important	79	91.86%
Important	25			
Not very Important	6	Not very Important	7	8.14%
Not Important	1			
Presential Meeting				
Very Important	18	Very Important	61	70.93%
Important	43			
Not very Important	24	Not very Important	25	29.07%
Not Important	1			

Question 07. Regarding testing artifacts, which are the ones you judge to be more appropriate for reuse?

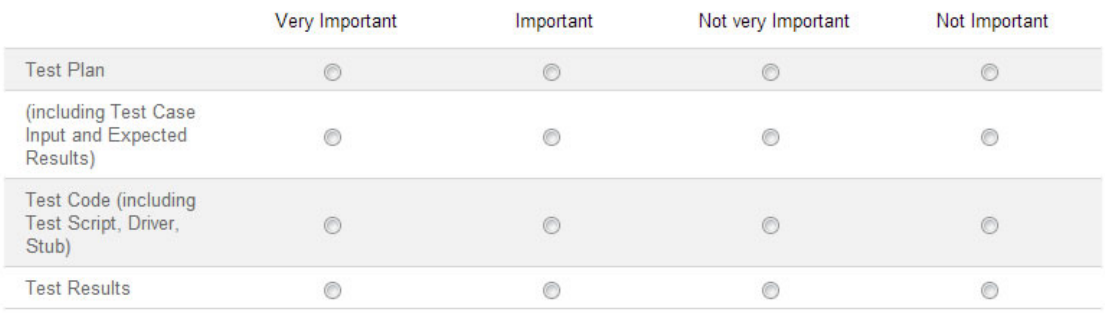

Figure A.3 - Question 07. Testing artifacts more appropriate for reuse

In this question, the test artifacts considered are showed in Figure A.3 were considered. Table A.7 presents the percentage of answers regarding the importance of reusing the test artifacts. In the opinion of the participants, "Test Plan" and "Test Case" artifacts are considered the most important artifacts for reuse in the software testing process, with percentages of 91.86% and 90.7%, respectively.

Table A.7 - Artifacts more appropriate for reuse

Test Plan				
Very Important	42	Very Important	79	91.86%
Important	37			
Not very Important	6	Not very Important	7	8.14%
Not Important	1			
Test Cases				
Very Important	46	Very Important	78	90.70%
Important	32			
Not very Important	7	Not very Important	8	1.16%
Not Important	1			
Test Code				
Very Important	30	Very Important	71	82.56%
Important	12			
Not very Important	25	Not very Important	15	31.40%
Not Important	3			
Test Results				
Very Important	25	Very Important	62	72.09%
Important	37			

Continues

Table A.7 - Conclusion

Not very Important	18	Not very Important	24	27,91%
Not Important	6			

Question 08. What is the purpose of applying KM in Software Testing? (Choose up to 2 options)

( ) Supporting decision-making process
( ) Reducing costs, time and effort
( ) competitive Advantages
( ) Improvement of intellectual capital
( ) Improving the quality of results
( ) Other

Table A.8 presents the percentage of answers in relation to the purposes of applying KM in software testing. "Improving the results quality" (27.91%) and "Reducing costs, time and effort" (25.6%) have the largest representativeness.

Table A.8 - Purpose of applying KM in Software Testing

Purpose	**Amount**	**Percentage**
Supporting decision-making process	34	19.77%
Reducing costs, time and effort	44	25.6%
Competitive Advantages	2	1.16%
Improvement of intellectual capital	35	20.35%
Improving the quality of results	48	27.91%
Other	9	5.23%

Question 09. What benefits KM can bring to software testing? (Choose up to 2 options)

( ) Selection and Application of better Testing Techniques
( ) Selecting the best test cases to perform
( ) Increasing the efficiency of the Testing Process
( ) Defining better Test Environments
( ) Other

Table A.9 presents "increasing the process testing efficiency" (40.7%) and the "selecting and applying better testing techniques" (approximately 33%) were the most representative expected benefits.

Table A.9 - Expected Benefits of applying KM in Software Testing

Expected Benefits	**Amount**	**Percentage**
Selection and Application of better Testing Techniques	57	33.14%
Selecting the best test cases to perform	17	9.88%
Increasing the efficiency of the Testing Process	70	40.7%
Defining better Test Environments	14	8.14%
Other	14	8.14%

APPENDIX B - Loading Existing Knowledge Items

This appendix presents details of the implementation of the load of data from the MantisBT and TestLink tools. As mentioned in section 5.3.1, in the context of ICAMMH Project, an integration scheme between TestLink and MantisBT was used. TestLink has the capability to integrate with MantisBT, allowing for a test case to be associated to a defect-related request. Thus, all incidents that were registered in MantisBT, as a defect-related request, were conditioned to the existence of a test case in TestLink. Thus, to load data into the knowledge repository of Testing KM Portal is necessary to work with the databases of these two tools.

The prototype was developed in the context of ICAMMH project. The data used as knowledge items of Test Case type corresponding to the data stored in the repositories of MantisBT and TestLink tools used in the project environment ICAMMH. As the project has been finalized, a copy of the data was obtained from the Project Coordination to make possible the development of a Testing KM Portal .

MantisBT and TestLink were created from a Structured Query Language (SQL) script containing a backup of the database ICAMMH project. Figure B.1 present tables of databases. The procedure to load and map data to Testing KM Portal is illustrated in Figure B.2.

The following describes each step of the procedure:

1. After creating the repositories, to load existing test cases, a functionality in Java was developed to connect with MantisBT and TestLink repositories. The MantisBT and TestLink database are in Mysql.

2. In this stage is performed the loading of all test cases and all its historical information from MantisBT and TestLink repositories. Routines were created to return the data from Mantis and TestLink with SQL scripts that makes the necessary relationships for each table used and their relationships. Figure B.3 presents an example of method that implements the burden of incident data from databases Mantis and TestLink and the corresponding SQL script.

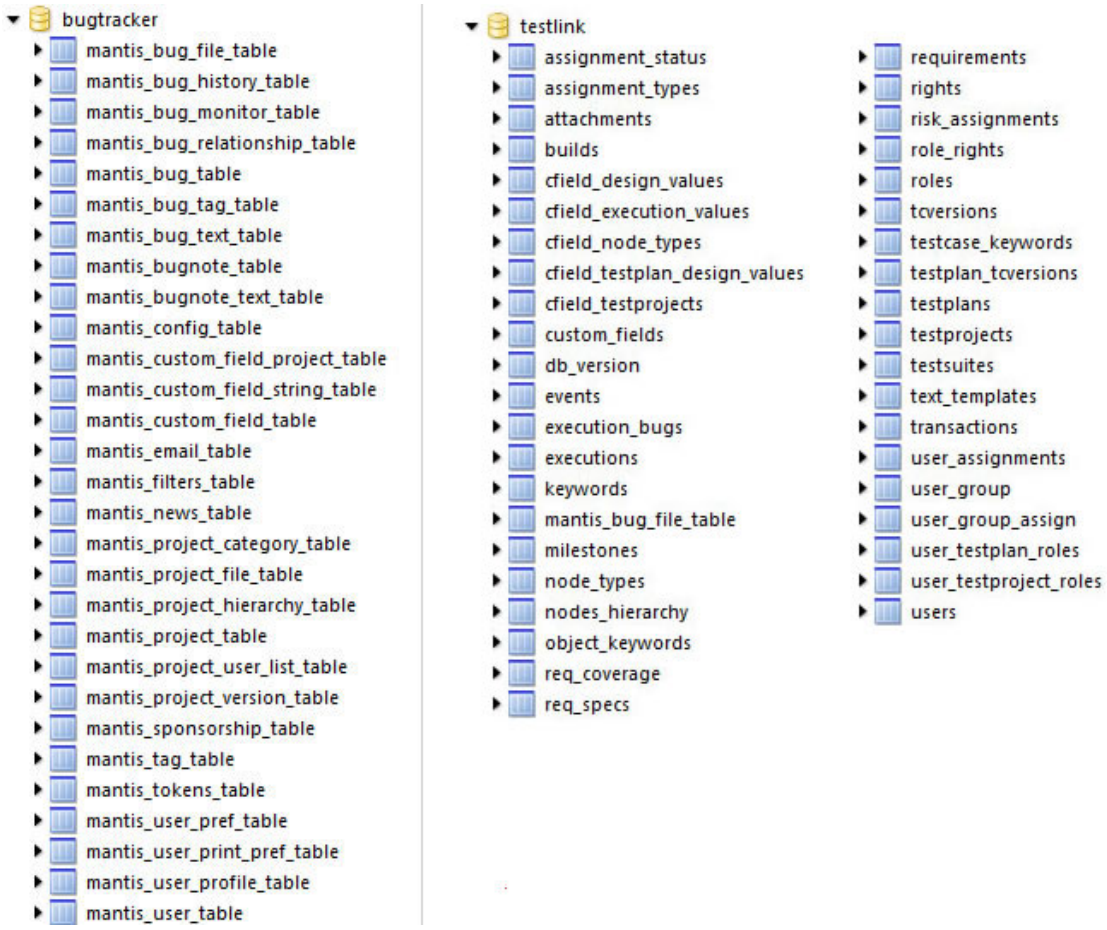

Figure B.1 - Tables of Mantis and TestLink

3. The data returned are mapped on objects that reflects the ROoST, that is, the data are in the data schema of Testing KM Portal repository. The following objects are structured: Human Resource, Knowledge Item, Test Case, Test Result, Incident and Issue. Each object contains their relationships and attributes, as shown in the conceptual model of Figure 5.13, Section 5.3.

A few information that was not in the repositories of data were registered manually trough the Testing Portal KM Portal, for example, information about the Project.

4. A connection to the repository Testing KM Portal is established. The portal database is in PostgreSQL.

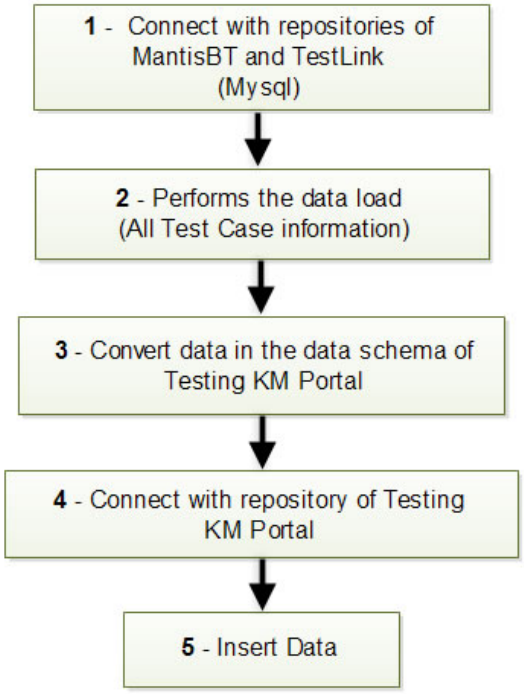

Figure B.2 - Process to loading knowledge items from Mantis and TestLink

```
public ArrayList<Incident> getIncidentsIssue() {

    ArrayList <Incident> incidentList = new ArrayList<Incident>();

    try {

        String SQL_STRING = "SELECT DISTINCT m.id AS mantis_bug_table, e.bug_id AS testlink_execution_bugs, m.category, "
                + "m.summary, t.description, m.status, tc.id, m.priority, m.resolution, m.reporter_id, m.date_submitted, m.reproducibility, "
                + "t.additional_information, m.target_version, m.fixed_in_version, m.severity, m.version "
            + "FROM mantis.mantis_bug_table m, mantis.mantis_bug_text_table t, testlink.execution_bugs e, "
                + "testlink.executions x, testlink.tcversions tc "
            + "WHERE t.id = m.bug_text_id and e.bug_id = m.id and e.execution_id = x.id and x.tcversion_id = tc.id";

        PreparedStatement statement = (PreparedStatement) connection.prepareStatement(SQL_STRING);
        ResultSet rs = (ResultSet) statement.executeQuery();

        while (rs.next()) {
            Incident incident = new Incident();
            incident.setIdIncident(rs.getInt("mantis_bug_table"));
            incident.setTargerVersion(rs.getString("target_version"));
            incident.setSubmissionDate(rs.getString("date_submitted"));
            incident.setSummary(rs.getString("summary"));
            incident.setDescription(rs.getString("description"));
            incident.setStateIncident(rs.getString("status"));
            incident.setPriority(rs.getString("priority"));
            incident.setFixedVersion(rs.getString("fixed_in_version"));
            incident.setSeverity(rs.getString("severity"));
            incident.setReproducibility(rs.getString("reproducibility"));
            incident.setStateResolution(rs.getString("resolution"));
            incident.setVersion(rs.getString("version"));

            incidentList.add(incident);
        }
         connection.close();
         ...
```

Figure B.3 - Loading the data corresponding to Incident

5. Finally the data is inserted into the Testing KM Portal repository. Figure B.4 presents an example of method that implements the insertion of

Incident data and the corresponding SQL script.

```
...
  while(itr.hasNext()) {
       Incident element = (Incident)itr.next();

  try {

      String sql = "insert into incidente (id, uuid, version, description, summary, targerversion, fixedversion, " +
              "stateresolution, reproducibility, severity, issue_id) values (?,?,?,?,?,?,?,?,?,?,?)";

      PreparedStatement stmt = (PreparedStatement) connection.prepareStatement(sql);

      stmt.setInt(1, element.getIdIncident());
      stmt.setInt(2, ' ');
      stmt.setInt(3,  ' ');
      stmt.setString(4, element.getDescription());
      stmt.setString(5, element.getSummary());
      stmt.setString(6, element.getTargerVersion());
      stmt.setString(7, element.getFixedVersion());
      stmt.setString(8, getStateResolution(element.getStateResolution())); //method that returns the corresponding Status Resolution
      stmt.setString(9, getReproducibilityDefect(element.getReproducibility())); //method that returns the corresponding Reproducibility
      stmt.setString(10, getSeverityDefects(element.getSeverity())); //method that returns the corresponding Severity
      stmt.setInt(11, element.getIdIncident()); //foreign key

      stmt.execute();
      stmt.close();

  } catch (SQLException e) {
      System.out.println(e.getMessage());
  }
...
```

Figure B.4 - Insertion of data corresponding to Incident