Journal of Software Engineering Research and Development, 2021, 9:12, doi: 10.5753/jserd.2021.1898

Development of an Ontology-based Approach for Knowledge Management in Software Testing: an Experience Report

Érica Ferreira de Souza [**Federal University of Technology - Paraná** | ***ericasouza@utfpr.edu.br***]
Ricardo de Almeida Falbo [**Federal University of Espírito Santo**]
Nandamudi L. Vijaykumar [**National Institute for Space Research** | ***vijay.nl@inpe.br***]
Katia R. Felizardo [**Federal University of Technology - Paraná** | ***katiascannavino@utfpr.edu.br***]
Giovani V. Meinerz [**Federal University of Technology - Paraná** | ***giovanimeinerz@utfpr.edu.br***]
Marcos S. Specimille [**Federal University of Espírito Santo** | ***marcosspecimille@gmail.com***]
Alexandre G. N. Coelho [**Federal University of Espírito Santo** | ***alexandregncoelho@gmail.com***]

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 large volumes of information are generated. Software testing is a knowledge-intensive process. Because of this, these organizations have shown a growing interest in Knowledge Management (KM) programs, which in turn support the improvement of testing procedures. KM emerges to manage testing knowledge, and, consequently, to improve software quality. However, there are only a few KM solutions supporting software testing. This paper reports experiences from the development of an approach, called Ontology-based Testing Knowledge Management (OntoT-KM), that aims to assist in launching KM initiatives in the software testing domain with the support of Knowledge Management Systems (KMSs). OntoT-KM provides a process guiding how to start applying KM in software testing. OntoT-KM is based on the findings of a systematic mapping on KM in software testing and the results of a survey with testing practitioners. Moreover, OntoT-KM considers the conceptualization established by a Reference Ontology on Software Testing (ROoST). As a proof of concept, OntoT-KM was applied to develop a KMS called Testing KM Portal (TKMP), which was evaluated in terms of usefulness, usability, and functional correctness. Results show that the developed KMS from OntoT-KM is a potential system for managing knowledge in software testing, so, the approach can guide KM initiatives in software testing.

Keywords: *Knowledge Management, Knowledge Management System, Software Testing, Testing Ontology*

1 Introduction

With the emergence of new technologies during the last decades, 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 life cycle, ensuring a well-managed process. Testing activities play an important role in assessing and achieving the quality of a software product (Souza, 2014).

Currently, software testing is considered a process consisting of activities, techniques, resources, and tools. Advances in technology and the emergence of increasingly critical applications also make testing a complex task. During software testing, large volumes of information are generated. 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 in an organization (Andrade et al., 2013; Souza, 2014). In this context, Knowledge Management (KM) emerges to manage testing knowledge, and, consequently, to improve software quality. KM can be defined as a set of organizational activities that must be performed systematically to acquire, organize, and sharing the different knowledge types in the organization (O'Leary and Studer, 2001). The adoption of principles of KM in software testing can help testers to promote reuse of knowledge, to support testing processes, and even to guide management decisions in organizations (Souza et al., 2015a).

Software testing, in general, can benefit from reusing test cases, testing techniques, lessons learned, and personal experiences, among others (Li and Zhang, 2012; Janjic and Atkinson, 2013; Souza et al., 2015a). To enable the reuse of testing knowledge, software organizations should be able to capture this knowledge and make it available to be shared with their teams. However, there are only a few KM solutions in the context of software testing (Souza et al., 2015a). The major problems in organizations regarding software testing knowledge are the low reuse rate of knowledge and barriers to knowledge transfer. This occurs because most of the testing knowledge in organizations is not explicit and it becomes difficult to articulate it (Souza et al., 2015a). On the other hand, implementing KM solutions, in general, is not an easy task. According to Storey and Barnett (2000), a large number of organizations are taking great interest in the idea of KM, but, these organizations are not familiar with how and where to start since they lack the proper guidance to implement KM. So, with an orientation on how to implement new KM solutions in the organization, or even with an existing solution that can be customized, becomes interesting for organizations since it is an opportunity for continued cost reduction, quality improvement, and reduction in software delivery (Rokunuzzaman and Choudhury, 2011).

Concerning technologies for KM, ontologies have been widely recognized as a key technology (Herrera and Martin-B, 2015). Ontologies can be used for establishing a common conceptualization to be used in the Knowledge Management System (KMS) to facilitate communication, search, storage, and representation of knowledge (O'Leary and Studer, 2001). However, only a few initiatives have used an ontology-based approach for KM in the software testing domain (Souza et al., 2015a).

This paper reports our experiences in developing an approach to assist in launching KM initiatives in the software testing domain with the support of KMSs. In this paper, we present OntoT-KM, an Ontology-based Testing Knowledge Management approach. OntoT-KM provides a process to apply KM in software testing. OntoT-KM considers the conceptualization established by a software testing ontology. A striking feature of OntoT-KM is to describe how a testing ontology can be used for guiding KM initiatives in software testing. The software testing ontology used in OntoT-KM is a Reference Ontology on Software Testing, called ROoST (Souza et al., 2017). ROoST was developed for establishing a common conceptualization about 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 making searching easier (Souza, 2014; Souza et al., 2017).

Lessons learned and experiences acquired in conducting this study are presented on two main fronts. Firstly, the OntoT-KM approach is presented to help software organizations to implement an initial KM solution in software testing. Subsequently, a prototype of a KMS was developed, called Testing KM Portal (TKMP), both as a proof of concept from the OntoT-KM approach, as well as understanding the needs of software development professionals in having a KMS in software testing ready and available for customization.

This research is an extension of a preliminary study published in (Souza et al., 2020). The extensions of this work are essentially threefold. First, we improved several sections to provide better research understanding through the inclusion of new text, extra depth in some paragraphs, and the inclusion of new figures and tables. Second, we analyzed the database created from ROoST using data mining techniques, to present the applicability of this type of research in the search for useful knowledge in knowledge repositories. Third, we improved the TKMP analysis by software engineering practitioners. We carried out an analysis separating the participants by professional position, such as professionals directly related to software development companies and professionals directly related to scientific research.

The main contributions of this research are the guidelines provided by OntoT-KM for guiding KM initiatives in software testing. These guidelines are supported not only by ROoST, but also from the findings of the mapping study Souza et al. (2015a) and the results of a survey with 86 testing practitioners. OntoT-KM was applied to develop TKMP, which was evaluated by test leaders of real projects in which it was applied. TKMP also was evaluated by 43 practitioners in terms of usefulness, usability, and functional correctness. Such evaluation was designed applying the *Goal, Question, Metric* (GQM) paradigm (Basili et al., 1994) and *Technology Acceptance Model* (TAM) (Davis, 1993).

The remainder of this study is structured as follows. Section 2 presents the main research concepts. Section 3 presents OntoT-KM. Section 4 presents the application of OntoT-KM and the evaluation results. Section 5 discusses related works. Finally, in Section 6, we present our final considerations.

2 Background

In this section, the main concepts of this study are discussed.

2.1 Software Testing

Software Testing consists of the dynamic Verification & Validation (V&V) activities of the behavior of a program on a finite set of test cases, against the expected behavior Abran et al. (2004). Testing activities are supported by a well-defined and controlled testing process (Abran et al., 2004). The process consists of several activities, namely (Abran et al., 2004), (Myers, 2004), (Black and Mitchell, 2011), (Mathur, 2012): test planning, test case design, test coding, test execution and test result. In the first activity, the testing should be planned, such as, the test environment for the project, scheduling testing activities, and planning for possible undesirable outcomes. Test planning is documented in a test plan. Then, in the test case design the test cases to be run are designed, documented, and then coded. During test execution, test code is run, producing results, which are then analyzed to determine whether test cases have been passed or failed.

The testing activities are performed at different levels. Unit testing focuses on testing each program unit or component. Integration testing takes place when such units are put together, aiming at ensuring that the interfaces among the components are defined and handled properly. Finally, system testing regards the behavior of the entire system (Abran et al., 2004), (Myers, 2004), (Black and Mitchell, 2011), (Mathur, 2012). In addition, many testing techniques are providing systematic guidelines for designing test cases, intending to make testing efforts more efficient and effective. Testing techniques can be classified, among others, as (Burnstein, 2003): white-box testing techniques, which are based on information about how the software has been designed and coded; black-box testing techniques, which generate test cases relying only on the input/output behavior, without the aid of the code that is under test; defect-based testing techniques, which aim at revealing categories of likely or predefined faults; and model-based testing techniques, which are based on models, such as Statecharts, finite state machines, and others.

One of the main characteristics of the software testing process is that it has a large intellectual capital component and can thus benefit from experiences gained from past projects (Souza et al., 2015a). During software testing, large volumes of information are processed and generated. So, it can be considered a knowledge-intensive process, making it necessary for automated support acquiring, processing, analyzing, and disseminating testing knowledge for reuse. In this context, Knowledge Management (KM) can be used (Souza et al., 2015a).

2.2 Knowledge Management

KM can be viewed as the development and leveraging of organizational knowledge to increase an organization's competitive advantage (Zack and Serino, 2000). In general, KM formally manages the increase of knowledge in organizations in order to facilitate its access and reuse, typically by using Information Systems (ISs) and KMSs (Herrera and Martin-B, 2015). In particular, KMSs aims at supporting organizations in knowledge management, in an automated way.

One issue in KMSs is how to represent knowledge. One alternative is ontologies (O'Leary and Studer, 2001) as they are considered a key technology for KM (Herrera and Martin-B, 2015), by defining the shared vocabulary to be used in the KMS facilitating knowledge communication, integration, search, storage, and representation. In ontology-based KMSs, ontologies are typically used to structure the content of knowledge items, to support knowledge search, retrieval, and personalization, serving as a basis for knowledge gathering, integration, and organization, and support knowledge visualization, among others.

KM has shown important benefits for software organizations. In Souza et al. (2015a), we performed a Systematic Mapping (SM) looking for studies presenting KM initiatives in software testing. An SM is a secondary study for an overview of a research area through the classification of the available evidence (Kitchenham and Charters, 2007). The main conclusions from this SM were: (i) There are few publications (only 15 studies were retrieved) addressing KM initiatives in software testing; (ii) The major problems that have motivated applying KM in software testing are low knowledge reuse rate and barriers in knowledge transfer; (iii) As a consequence, knowledge reuse and organizational learning are the main purposes for managing software testing knowledge; (iv) There is a great concern with both explicit and tacit knowledge; (v) Reuse of test cases is the perspective that has received more attention; (vi) KMSs are used in almost all initiatives (11 of the 15 studies); and (vii) Different technologies have been used to implement those KMSs, such as conventional technologies (databases, intranets, and Internet), yellow pages (or knowledge maps), recommendation systems, data warehouse, and ontologies.

In particular, one finding drew our attention: only two studies, actually, used ontologies in a KM initiative applied to software testing (Liu et al., 2009; Li and Zhang, 2012). This seems to be a contradiction, since, as pointed out by Staab et al. (2001), ontologies are the glue that binds KM activities together, allowing a content-oriented view of KM. One possible explanation for this low number of studies is the fact that developing an ontology is a hard task, especially in complex domains, as is the case of software testing (Souza et al., 2015a).

Based on the findings of the SM, we decided to perform a Systematic Literature Review (SLR) looking for ontologies on the software testing domain in the literature (Souza et al., 2013). An SLR also is a secondary study that uses a well-defined process to identify available evidence (Kitchenham and Charters, 2007). From this SLR, 12 ontologies addressing this domain were identified. As the main findings, it is possible to highlight (Souza et al., 2013): (i) Most ontologies have limited coverage; (ii) The studies do not discuss how the ontologies were evaluated; (iii) None of the analyzed testing ontologies is truly a reference 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; and, finally, (iv) 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, none of the analyzed ontologies is grounded in foundational ontologies. This motivated us to build ROoST, a Reference Ontology on Software Testing (Souza et al., 2017). ROoST was developed for establishing a common conceptualization of the software testing domain.

2.3 ROoST

ROoST is presented very briefly here since it is not the scope of this paper to present the entire ontology. Details of the ontology can be found in (Souza et al., 2017). Since the testing domain is complex, ROoST was developed in a modular way, comprising four modules (sub-ontologies): (i) **Software Testing Process and Activities** representing the testing process and the main activities that comprise it, namely test Planning, test case design, test coding, test execution, and analysis of the test results; (ii) **Testing Artifacts** focusing on the artifacts used and produced by the testing activities; (iii) **Techniques for Test Case Design** looking at testing techniques, such as black-box, white-box, defect-based, and model-based testing techniques; and (iv) **Software Testing Environment** addressing the main components of the testing environment, including test hardware resources, test software resources, and human resources.

In order to develop ROoST, the Systematic Approach for Building Ontologies (SABiO) (Falbo, 2014) was adopted. SABiO method incorporates best practices commonly adopted in Software Engineering and Ontology Engineering and addresses the design and coding of operational ontologies. Furthermore, ROoST has been developed by reusing and extending ontology patterns from the Software Process Ontology Pattern Language (SP-OPL) (Falbo et al., 2013) and the Enterprise-Ontology Pattern Language (E-OPL) (Falbo et al., 2014). An Ontology Pattern Language (OPL) is a network of interconnected domain-related ontology patterns that provides holistic support for solving ontology development problems for a specific domain (Falbo et al., 2013). More recently, ROoST has been integrated into the Software Engineering Ontology Network (SEON) (Ruy et al., 2016). The full model of ROoST is available at *http://dev.nemo.inf.ufes.br/seon/*.

Given the size of ROoST, Figure 1 presents only its *Testing Process and Activities* sub-ontology. Concepts reused from Software Process Ontology are shown in gray. Specific Concepts are shown in white. Some of the main concepts of this sub-ontology are also presented below. More specific details of ROoST's *Testing Process and Activities* sub-ontology can be found at (Souza et al., 2017) and SEON Network[1].

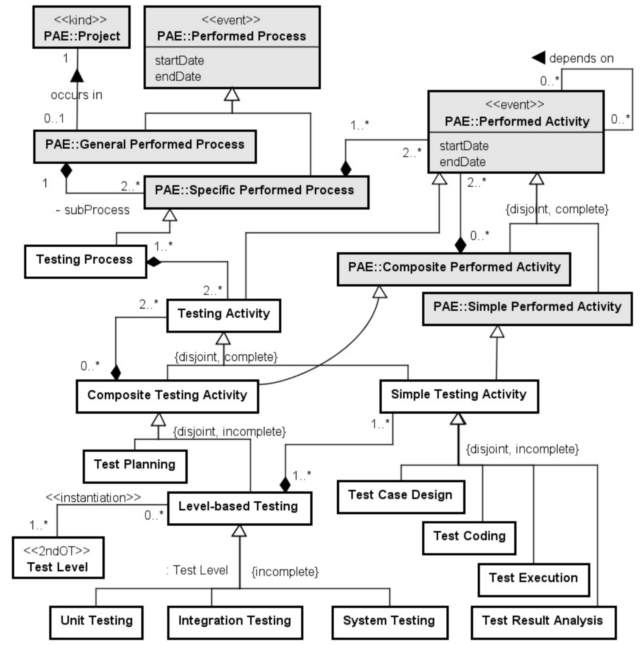

Figure 1. ROoST's Testing Process and Activities sub-ontology

In this sub-ontology, the Process and Activity Execution (PAE) pattern was reused. PAE concepts were extended to the testing domain, as shown in Figure 1. *Testing Process* is a subtype of *Specific Performed Process*, since a testing process occurs in the context of the entire software process (*General Performed Process*) of a *Project*. A testing process, in turn, is composed of testing activities, and thus *Testing Activity* is considered a subtype of *Performed Activity*. Similarly to *Performed Activity*, *Testing Activity* can be further divided into *Composite* and *Simple Testing Activity*. In PAE pattern, *Specific Performed Processes* are composed of two or more *Performed Activities*. A *Performed Activity*, analogously, can be simple or composite. A *Composite Performed Activity* is composed of other *Performed Activities*; a *Simple Performed Activity* cannot be decomposed into smaller activities (Falbo et al., 2013).

Besides specializing concepts, relationships were also specialized from PAE. For instance, in PAE, there is a whole-part relationship between *Specific Performed Process* and *Performed Activity*. The whole-part relationship between *Testing Process* and *Testing Activity* 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.

Regarding the testing process activities, *Test Planning* and *Level-based Testing* are *Composite Performed Activities*. Although not shown in Figure 1, 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. *Level-based Testing* comprises *Test Case Design*, *Test Coding*, *Test Execution* and *Test Result Analysis*, which are considered *Simple Performed Testing Activities*.

Considering the test levels, *Level-based Testing* groups simple testing activities according to the *Test Level* to which they are related. Thus, *Level-based Testing* is specialized according to the instances of *Test Level*, a second-order type, whose instances partition Level-based Testing in more specific types of testing activities. In Figure 1, the three most cited testing levels in the literature are made explicit: *Unit Testing*, *Integration Testing* and *System Testing*. However, there may be others, such as *Regression Testing*.

Regarding testing stakeholders, the *Test Manager* is responsible for performing *Test Planning* activities. *Test Manager* also participates in *Test Result Analysis* activities. *Test Case Designer* participates in *Test Planning* activities, and she is in charge of performing *Test Case Design* and *Test Result Analysis* activities. Finally, the *Tester* is responsible for performing *Test Coding* and *Test Execution*.

With respect to testing artifacts, *Test Planning* produces a *Test Plan*, which is used by *Level-based Testing* activities. *Test Case Design* uses several artifacts as *Test Case Design Inputs* and applies *Testing Techniques* for developing *Test Cases*. During *Test Coding*, *Test Code* is produced, implementing a *Test Case*. During *Test Execution*, *Test Cases* are executed by running the *Code to Be Tested* and the *Test Code*, producing *Test Results*. Finally, in a *Test Result Analysis* activity, *Test Results* are analyzed and the findings are reported in a *Test Analysis Report*.

3 OntoT-KM

Given the applicability of KM to improve software testing processes, we developed OntoT-KM for assisting companies that want to create their solutions for KM initiatives in dynamic software testing, supported by a KMS. OntoT-KM consists of a process and a set of guidelines for implementing a KMS in software testing organizations. OntoT-KM is supported by ROoST, in particular, to structure the KMS knowledge repository. Moreover, OntoT-KM guidelines are based on the findings of the SM presented in (Souza et al., 2015a), and the results of a survey performed with testing practitioners presented in (Souza et al., 2015b).

The OntoT-KM process comprises the following steps: (i) Diagnose the Organization's Testing Process; (ii) Establish the Testing KM Scope; (iii) Develop a Testing KMS; (iv) Load Existing Knowledge Items; and (v) Evaluate the Testing KMS. Figure 2 presents OntoT-KM process as a UML activity diagram. As this figure shows, ROoST is used to support steps (i) and (iii). Steps (i) and (ii), shown in another color in Figure 2, are considered optional. Following, each process step is presented, describing the main guidelines that apply.

Step 1: Diagnose the Current State of the Organization's Testing Process

The first step of the OntoT-KM process is to make a diagnosis of the current state of the organization's testing process. It 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. This step may become optional given the organization's maturity level. This step is an important step for organizations with low maturity. Once identifying the knowledge items, organizations can then proceed to

[1] *http://dev.nemo.inf.ufes.br/seon/*

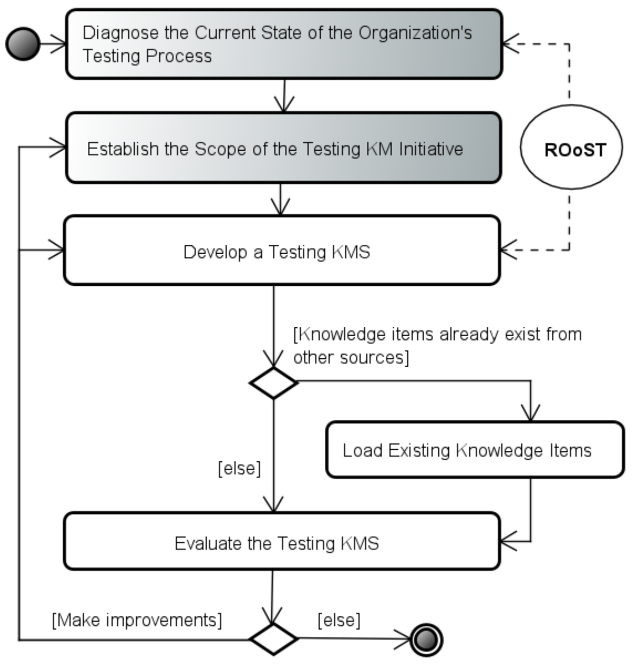
Diagnose the Current State of the Organization's Testing Process
Establish the Scope of the Testing KM Initiative
ROoST
Develop a Testing KMS
[Knowledge items already exist from other sources]
Load Existing Knowledge Items
[else]
Evaluate the Testing KMS
[Make improvements]
[else]

Figure 2. OntoT-KM Process

manage them.

This step may be accomplished by surveys with questionnaires and/or interviews. This activity should consider the entire current state of software testing in the organization concerning, at least, the following aspects: the adopted testing process, the activities of the candidate testing processes to be targeted by the KM initiative, the artifacts produced during this process, the testing techniques applied, the test levels contemplated by the process, and the test environments adopted by the organization's software projects. Aspects related to KM should also be investigated, such as the current KM practices applied in the testing process, the organization's purpose of applying KM to software testing, problems related to testing knowledge in the organization, among others. ROoST can be used in this step as the common vocabulary for supporting the analysis of the current status, as well as to formulate the questions to be used in the questionnaires and/or interviews with the organization's testing practitioners. The results of these questionnaires/interviews should be used as guidelines for the next step (establish scope).

For this step, we suggest asking, at least, the following questions to the testing practitioners participating in the diagnosis:

- *What are the testing activities that comprise the organization's testing process?* Evaluate the answers considering the consensual activities considered in ROoST (Test Planning, Test Case Design, Test Coding, Test Execution, and Test Result Analysis), and consider the possibility of improving the organization's testing process by aligning it to the testing process captured by ROoST.
- *In which activities of the Testing Process is KM more useful?* The activities pointed out in the previous step should be the ones considered here as possible answers.
- *What are the testing levels in which the organization performs tests?* Testing activities can be performed at different levels. Taking ROoST as a basis, consider at least the following levels: unit testing, integration testing, and system testing. However, if the organization tests software at other levels, these should be considered.
- *In which testing level is KM more useful?* The testing levels pointed out in the previous step should be the ones considered here as possible answers.
- *Which resources do you consider more important to have the knowledge available about them when defining the testing environment?* According to ROoST, the possible answers that are considered for this question are the following types of resources: hardware, software, and human.
- *Concerning tacit and explicit knowledge, what are the types of knowledge you consider to be more important to manage during the software testing process?* Testing practitioners tend to consider both useful, but we need to evaluate which one is more important and which is easier to implement. In general, for organizations starting a KM initiative in software testing, explicit knowledge is easier to handle. In particular, test cases highlight the most important artifacts to be managed as a knowledge item, as pointed in SM (Souza et al., 2015a) and the survey (Souza et al., 2015b).
- *What is the purpose of applying KM in Software Testing? What benefits can KM bring to software testing?* This question captures the feeling of testing practitioners regarding why and how an organization can benefit by applying KM to software testing.

Step 2: Establish the Scope of the Testing KM Initiative

Once the diagnosis of the status of the testing process has been carried out, the next step is to establish the KM scope. Similarly, as in the case of step 1, this step may also be considered optional if the organizations already know their needs. For the KM scope, it is necessary to familiarize oneself with the organization's needs. The organization must define the testing process activities that are to be supported, and knowledge types to be managed.

A major challenge for organizations is to know which knowledge is useful, and thus identify potential knowledge items among the several knowledge assets generated in the testing process. Results from step 1 should be used here. In addition, it is suggested that organizations start with small KM initiatives.

As a general guideline, we recommend considering the survey results that we performed (Souza et al., 2015b). From this survey, both test case design and test planning were considered the most important testing activities to be supported by KM practices, and capturing as knowledge items their main outcomes, namely: test case and test plan.

When considering test cases as knowledge items, it is necessary to build an appropriate infrastructure that allows for the analysis, storage, and retrieval of existing test cases. This structure can be achieved from the OntoTKM approach. In the reuse of this knowledge item, for example, the test reuse system may be able to cover a variety of search scenarios in order to assist its users in different situations. The search engine enables searching for test cases by informed parameters, for example, test levels or testing techniques. The returned test cases can be reused in similar scenarios. According to Werner (2014), reusable tests that have been written for a similar scenario are likely to help to better understand how a previously created similar system works. In addition, by reusing the knowledge contained in existing tests, developers can benefit from the knowledge that others have invested in developing them. These tests can help to gain better insights into how a particular kind of component should behave.

Regarding test planning, the survey led to the selection of testing techniques and the definition of the testing environment as the most important tasks to be supported by a KM initiative. Concerning knowledge about the testing environment, managing knowledge about human resources and software resources is pointed out as the most promising approach. Regarding knowledge about human resources, this impression is corroborated by the SM (Souza et al., 2015a), where yellow pages and knowledge maps appear in various initiatives.

Other knowledge items related to making tacit knowledge explicit can also be considered in KM scope, namely:

- **Lessons Learned (LL):** 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, points of view, decisions, among others. In addition, LL can also be classified as informative, success, or failure lessons. Informative LLs explain how to proceed in a given situation; lessons of success provide examples of problems that were solved positively; and failure lessons provide examples of negative responses to attempt to solve a problem and potential ways to cope up with the situation (O'Leary, 1998).
- **Knowledge regarding discussion:** Discussion among the organization's members may be submitted as knowledge items. Tools to support discussion among the organization's members, such as discussion forums, have been fundamental in KM environments (Fischer and Ostwald, 2001). Discussion forums become important tools for knowledge management for the following reasons: (i) very useful knowledge can be generated and captured during discussions (Falbo et al., 2004), and (ii) a major challenge of KM is to convert tacit knowledge into explicit knowledge (Nonaka and Krogh, 2009; Davenport and Prusak, 2000).

Step 3: Develop a Testing KMS

This phase concerns developing a KMS to support the KM initiative and comprises the main activities for developing systems, in general: requirements specification, design, implementation, and testing.

Requirements must be elicited and specified. Functional requirements may be created from use case models, class diagrams, and state diagrams to model the behavior of knowledge items throughout their existence in the KMS. 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 for the KMS, and thus as the basis for structuring the testing knowledge repository. Specific information (attributes of the classes in the conceptual model) should be identified, taking the characteristics of the organization's testing artifacts into account, and, most importantly, information that is available in the tools used for supporting the testing process.

Furthermore, interoperability issues should also be analyzed. Ideally, software tools that are part of the test environment should be integrated with the Testing KMS 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.

Another key point is to define the KM process activities that are to be supported by the Testing KMS. We recommend providing support to the following typical activities of a KM process: creating knowledge items, evaluating knowledge items before making them available, searching knowledge items, assessing the usefulness of available knowledge items, and maintaining the knowledge repository.

During the design of Testing KMS, developers should consider the platform in which the system is to be built, and non-functional requirements should be addressed. Several technologies can be used, including those that are commonly considered in KM solutions like content management systems, document management systems, and wiki, as well as those considered intelligent KM solutions, such as knowledge-based and expert systems, reasoners, and semantic wikis. Once designed, the KMS should be coded and tested.

Step 4: 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 must manage test cases, existing test cases can be imported to the Testing KMS. The existing knowledge items should be reengineered to ensure conformance with the knowledge repository structure. Knowledge items can be registered manually in the Testing KMS or mechanisms for loading and reengineering these knowledge items can be built to automate the loading process.

Once a knowledge repository can be created and populated, data mining can be explored. The knowledge repository can contain useful hidden information (knowledge) of major relevance to the business, so, 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 (KDD), process knowledge data structuring (Fayyad et al., 1996). 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., 1996).

Mining stored data, in large databases to discover potential information and knowledge, has been a popular topic in database research. Data mining is a technology to obtain information and valuable knowledge (Yun et al., 2003). According to Basili and Rombach (1991), the quality of software development can be improved by reusing acquired experiences, rather than starting from scratch. Therefore, as a result of applying KDD, another knowledge item type can be considered: **Mined items**. Mined items can be provided by a mining process in a KM database and can identify relationships that are not apparent, facilitating decision-making. Furthermore, identifying behavior patterns in data stored in knowledge bases can help the organization to reuse and share the knowledge acquired in previous projects.

Step 5: Evaluate the Testing KMS.

Evaluation should be done to determine if the Testing KMS meets the expectations. Improvements can be carried out, implying a return to the previous steps. A suggestion to evaluate the testing KMS is to analyze some quality characteristics, such as usefulness, usability, and functional correctness. To do that, two models can be considered: GQM (Basili et al., 1994) and TAM (Davis, 1993).

GQM is a measurement model, organized into three levels. In the first level (conceptual level), the study goals should be defined. The second level (operational level) refers to a set of questions that should be defined to characterize the evaluation or the accomplishment of a specific goal. Finally, in the last level (quantitative level), a set of metrics should be associated with questions, to answer them measurably. The result of applying the GQM approach is the specification of a measurement system targeting a particular set of issues and a set of rules for interpreting measurement data (Basili et al., 1994). GQM is useful because it facilitates identifying not only the precise measures required but also the reasons why the data are being collected (Park et al., 1997).

TAM determines the acceptance of a given technology by users, considering two-factor analysis: usefulness and usability. When evaluating these two factors, it is possible to map the users' acceptance of new technology.

Usefulness refers to how much the user realizes that certain technology is useful to her in terms of productivity increase. According to ISO/IEC 25010 (ISO/IEC, 2011), usefulness is the "degree to which a user is satisfied with their perceived achievement of pragmatic goals, including the results of use and the consequences of use". In this standard, usefulness is part of the quality in use model.

The perception of usability refers to the effort reduction that the user achieves when using a given technology instead of using other alternatives (Davis, 1993). In ISO/IEC 25010 (ISO/IEC, 2011), usability refers to the "degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use". It is a quality characteristic of the product quality model, but for consistency with its established meaning, it is also defined as a subset of the quality in use model.

In this work, we also decided to evaluate another quality characteristic: functional correctness, a sub characteristic of functional suitability in the ISO/IEC 25010 product quality model. According to ISO/IEC (2011), functional correctness is the "degree to which a product or system provides the correct results with the needed degree of precision".

4 Applying OntoT-KM

Our experience in developing the OntoT-KM approach has two fronts. First, we introduce the approach, and then we create a prototype of a KMS based on OntoT-KM that allows us to evaluate the approach, as well as obtain the opinion of software professionals in having a KMS in software testing ready and available for customization. Following, the KMS development based on OntoT-KM and all processes of the evaluations are presented.

To evaluate the OntoT-KM approach, we applied it to build a prototype of a KMS for managing software testing knowledge, called Testing Knowledge Management Portal (TKMP). The resulting system was populated with data from two real projects and different evaluations were conducted. The projects were (Souza, 2014): (i) Amazon Integration and Cooperation for Modernization of Hydrological Monitoring (ICAMMH) Project; and (ii) On-Board Data Handling (OBDH) software inside the Inertial Systems for Aerospace Application (SIA) Project.

ICAMMH Project was a collaboration involving the Brazilian Aeronautics Institute of Technology and the Brazilian National Water Agency, supported by the Brazilian Financial Foundation for Projects - FINEP. The project developed a pilot system for modernization and integration of telemetry points collected from hydrological data, as a basis for managing water resources in the Amazon region. The second project is devoted to developing software for the onboard computer of the SIA Project, which is a computational system for OBDH (On-Board Data Handling) to Attitude and Orbit Control (AOC) of satellites that can be adapted for future space applications at the National Institute for Space Research (INPE). The first version of the OBDH software was in the testing phase when this work was being done. The final version of this software aims at adding all the functionalities of OBDH of a 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). At the time we were carrying out this research, ICAMMH Project had already been finalized and the SIA project was in its early stage.

4.1 Diagnose the Current State of the Organization's Testing Process

As ICAMMH Project has already been finalized and the testing activities of the SIA project were only in the very initial phase, it was not possible to run the diagnostic step. This step was replaced by the findings from the survey with 86 testing practitioners we performed (Souza et al., 2015b). Out of these 86 participants, some are also team members and leaders of the ICAMMH and SIA project.

The survey's purpose was to identify which is the most appropriate scenario in the software testing domain, from the point of view of testing stakeholders, for starting a KM initiative. The survey presents questions that addressed aspects considered both in the conceptualization of ROoST and by the SM presented in (Souza et al., 2015a), as shown in Table

1. Furthermore, managing testing knowledge is not an easy task, and thus it is better to start with a small-scale initiative. Thus, firstly, it is necessary to identify essential knowledge items of a sub-topic of software testing to be dealt with in the KMS.

From the survey results, the following conclusions are considered: (i) the participants identified test case design and test planning as being the activities in which KM would be most useful. Therefore, test cases and test plans are considered the most useful artifacts to be reused; (ii) explicit knowledge was considered more important than tacit knowledge. Explicit knowledge represents the objective and rational knowledge that can be documented, and thus it can be accessed by many (Nonaka and Takeuchi, 1997). On the other hand, the tacit knowledge is the subjective and experience-based knowledge and typically remains only in people's minds (Nonaka and Takeuchi, 1997); (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, and reducing cost, time and effort spent in a software project.

4.2 Establish 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 case** the main artifact to be managed. All relevant information for designing test cases had to be considered in the scope of the TKMP development. Thus, concepts related to test cases in ROoST were also considered in the scope of the initiative, namely: *Test Case Input, Expected Result, Test Result, Test Code, Test Case Designer* and *Testing Technique*.

Besides the test cases as the main artifacts to be managed, LL and Knowledge regarding discussion were also considered in the scope of TKMP.

These two types of knowledge items were considered in the scope of TKMP since survey participants pointed out individual experiences and communication between test team members as the types of tacit knowledge with more significant importance to generate explicit knowledge items. In addition, meetings with the project leaders from ICAMMH and SIA projects also helped to reach this scope.

Still, concerning tacit knowledge, we decided that TKMP should also include a Yellow Page System since survey participants pointed out human resources as the most useful resource to be managed and test case designers are in the scope of this KM initiative.

Finally, we also decided to apply KDD for discovering useful knowledge from existing data and identifying the mined items. As presented in Section 3, Step 4, different mining methods can be used in the identification of relevant information in large volumes of data. In this project, for creating the mined items, the method of association rule was used. The association rule method identifies patterns of behavior in the set of data that often occur jointly in the database and model rules from these sets. The association rules, when applied to a data set, allow finding rules of the type of $X \rightarrow Y$, i.e., transactions of the database which contain X tend to contain Y. The method of Rule Association was used along with the Apriori algorithm (Agrawal and Srikant, 1994; Witten et al., 2005). Apriori algorithm is the best known in rule discovery methods (Agrawal and Srikant, 1994).

4.3 Develop a Testing KMS

Considering the scope defined in the previous activity, TKMP was developed. The specification of the main requirements was developed, such as the use case diagram and class diagram conceptual model. Figure 3 shows a partial use case diagram describing the main functionalities of TKMP and actors. The use cases in gray are general, in the sense that they apply to manage software engineering knowledge items of different nature. Use cases in white represent testing-specific features. The *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. Next, the use cases shown in Figure 3 are briefly described.

- **Create Knowledge Item:** This use case allows developers to create a knowledge item.
- **Create Discussion-related Knowledge Item:** This use case allows developers to register a Discussion-related Knowledge Item.
- **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 Incident:** This use case allows the developer to report an incident related to a test result.
- **Include Issue:** This use case allows the developer to register an issue related to an incident.
- **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 per informed parameters.

Table 1. Relationships between the Survey Questions (SQ) and the Research Questions (RQ) from the mapping study and ROoST

Survey Question	***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 when defining the test environment?	**ROoST**: Testing Environment sub-ontology
SQ4. In which testing level is KM more useful?	**ROoST**: Testing Process and Activities 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 software testing?
SQ6. Regarding the types of knowledge items listed below, indicate the 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 benefits and problems reported regarding applying KM in software testing?

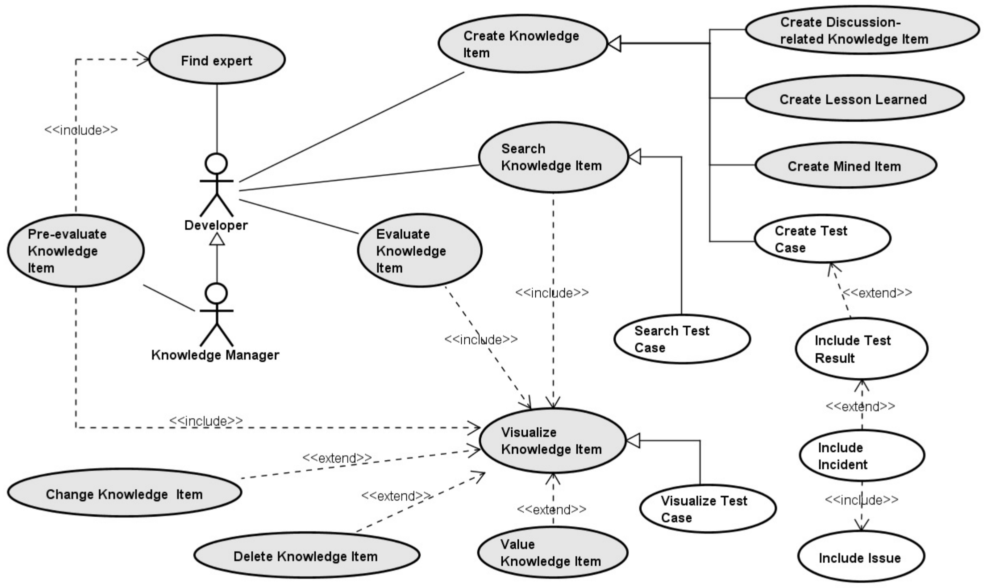

Figure 3. Functionalities of TKMP

- **Search Test Case:** This use case allows the developer to search for test cases per 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 4 shows a partial conceptual model of TKMP. This model focuses on Knowledge Items, on test cases. Classes in gray are derived from ROoST, i.e., the ROoST conceptual model was used as the starting point for specifying TKMP, mainly to structure its knowledge repository regarding the software testing notions. Information from the software tools that compose the testing environment of the ICAMMH Project was used as the basis for identifying attributes and enumerated types, to specify TKMP in detail. These tools are TestLink[2], a web-based test management system, and MantisBT[3], a bug tracking system.

[2]http://testlink.org/
[3]http://www.mantisbt.org/

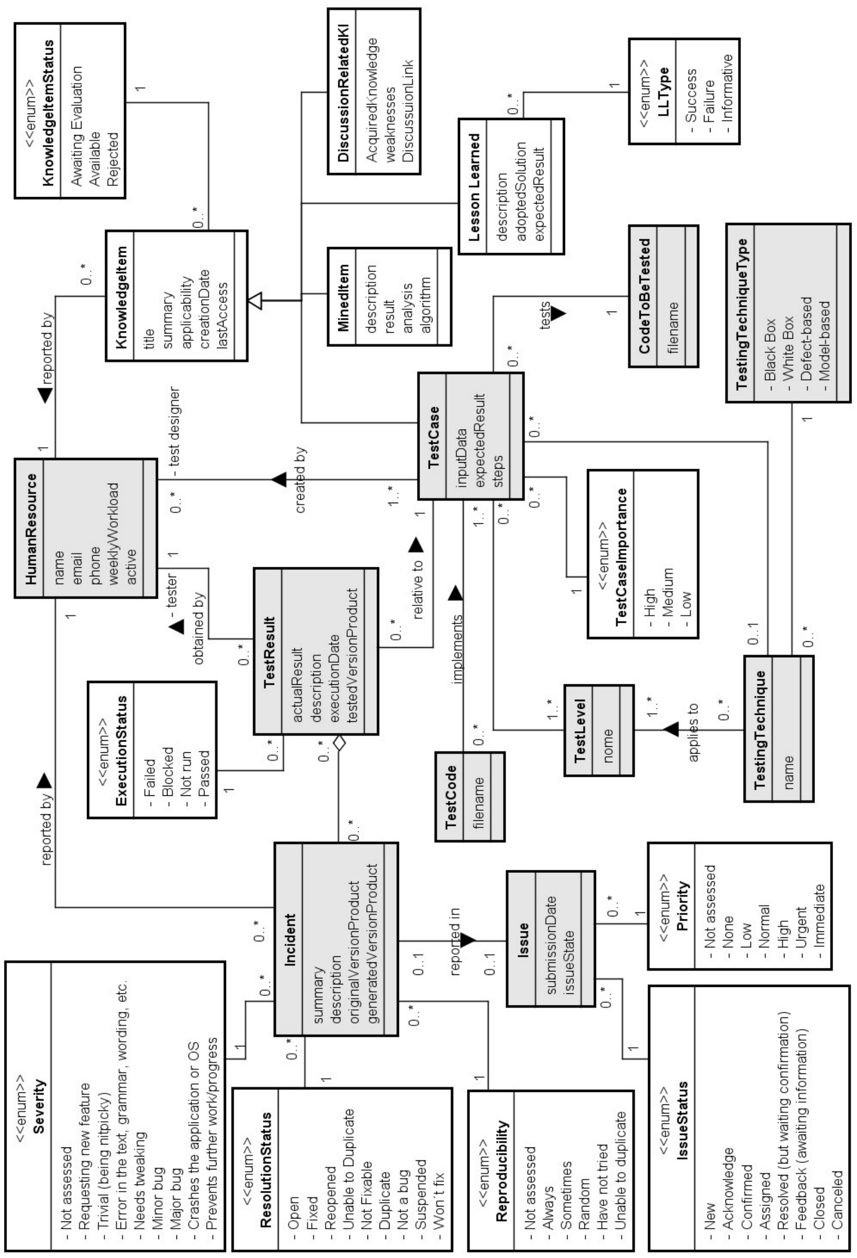

Figure 4. Conceptual model of TKMP.

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

TKMP project and requirements specifications are currently available at *https://cutt.ly/KyBOLUn*.

4.4 Load Existing Knowledge Items

Once TKMP was developed, previous existing knowledge items in the two projects were loaded to the knowledge repository. Initially, TKMP's knowledge repository was populated with 1568 test cases extracted from ICAMMH Project. Next, other test cases from the SIA Project were also inserted in TKMP's knowledge repository using TKMP's functionalities.

In the context of the ICAMMH Project, test case related information was stored both in TestLink and MantisBT. Each one of these tools has its data repository, implemented in different ways, demanding analysis of the structure of each one to load the data. Moreover, each tool has its terminology to represent the manipulated data, i.e., different terms are used to represent the same concept. Thus, to load existing test cases, a feature was developed to connect and get data from the repositories of MantisBT and TestLink, and then to convert them into objects (instances) of the data schema of TKMP. ROoST was used for mapping concepts from the involved tools. This procedure is illustrated in Figure 5.

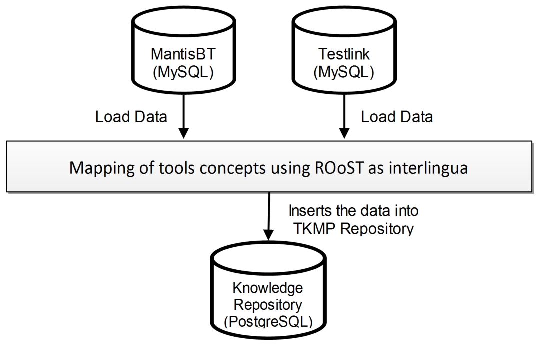
MantisBT (MySQL)
Testlink (MySQL)
Load Data
Load Data
Mapping of tools concepts using ROoST as interlingua
Inserts the data into TKMP Repository
Knowledge Repository (PostgreSQL)

Figure 5. Loading existing knowledge items

In this step, we decided to mine the stored knowledge items, since mined items also were considered as scopes of KM initiatives in Software Testing (see Section 4.2). Data mining was performed on ICAMMH data. To create the mined items, the method of Rule Association was used. 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 Waikato Environment for Knowledge Analysis (WEKA) tool 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 2 presents the first 20 returned and 8 attributes considered in this data mining, corresponding to classes: Human Resource, Test Case, Incident, Issue.

After loading the data set, the Apriori algorithm was executed using the WEKA tool. WEKA returns the most important 10 associations. This number can be changed in the algorithm settings. The listing in Table 3 shows the results of the associations that were found.

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 with all the other rules, one realizes that there are consistencies among associations that were presented.

About the associations returned with the TKMP data, no irregularities were detected. In this case, it is concluded that the classes used to generate associations have the correct registration patterns by the project members. However, more classes could be incorporated into the associations to allow more analyses of the data. Furthermore, other mining algorithms could be used. By using association rules combined with other mining methods one could detect behaviors not seen by the naked eye, for example, to notice or register a certain type of defect tends to appear when changing a certain software component or the severity of the test case is always major when they are related to a particular module. Behaviors like these could help the responsible expert in project decisions related to the tests being conducted.

For the registration of a knowledge item of the mined item type in TKMP, generic information about that item was considered given the diversity of methods and algorithms that exist in data mining. In the conceptual model of TKMP (Figure 4), the *MinedItem* table shows the attributes that are available for the registration of a mined item. The attributes are: Description, Algorithm, Result, and Analysis.

4.5 Evaluate the Testing KMS

Although TKMP is still considered a prototype, built as a proof of concept for the OntoT-KM approach, we decided to conduct different evaluations for this KMS in order to get the feeling of software professionals in having a KMS available for customization.

TKMP went through a preliminary evaluation in two steps. Firstly, TKMP was evaluated by the leaders of the two projects, ICAMMH and SIA. Secondly, TKMP was made available on the Web, and software engineering practitioners were invited to use it and then to answer a questionnaire to give feedback in terms of usefulness, usability, and functional correctness.

4.5.1 Evaluation with the project leaders

Once TKMP's knowledge repository was populated with data from the two real projects (ICAMMH and SIA), demonstrations with the data obtained from the projects were made and the leaders were requested to use and analyze the portal. Then, we interviewed in order to collect opinions and/or impressions of the leaders about the TKMP. The interview was conducted in an unstructured manner and anonymously. This configuration for the interview allowed information to emerge more freely. We began the interview by considering three open questions to serve as a guide: "*What is the perceived usefulness of TKMP?*", "*Do you think it's easy to learn to use TKMP?*" and "*Do you notice inconsistencies when using TKMP?*". Open questions allowed respondents a wide range of answers and diverse discussions about the tool. Some of the leaders' comments on the TKMP are presented below.

The leaders of both projects stressed the importance of such a system to better support the software testing processes. Positive responses were presented by the leaders to the TKMP in terms of usefulness, usability, and inconsistencies.

With respect to ICAMMH Project, the leader observed that there was always a great loss of knowledge due to the turnover rate of the team members. In her words, "a KMS such as TKMP would be indeed beneficial for finding similar test cases to be reused in the design of new ones to other similar situations in different modules and future projects".

With respect to the SIA Project, the leader's evaluation was that TKMP would be very important for dealing with critical systems. However, he pointed out that a challenge would be to change team members' culture because many

Table 2. 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
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

Table 3. Results of the associations

Rule	Associations
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)

times the team is not ready or does not accept new concepts, tools, and ideas.

4.5.2 Evaluation by software engineering practitioners

TKMP was also evaluated by 43 practitioners in Software Engineering, and it was based on GQM, TAM, and functional correctness. The evaluation based on the GQM paradigm involved four steps: Planning, Definition, Data Collection, and Interpretation.

(I) Planning and Definition. At GQM's conceptual level, measurement goals should be defined. We identified three goals for this evaluation, and from these goals, at the operational level, we defined seven questions, as Table 4 shows. Finally, at the quantitative level, we defined metrics associated with the questions, in order to answer them measurably. For each question, as Table 5 shows, we defined five metrics, each one aiming at computing the number of participants that strongly disagree (MG.Q.1), disagree (MG.Q.2), neither agree nor disagree (MG.Q.3), agree (MG.Q.4), or strongly agree (MG.Q.5) with a statement corresponding to the question. Figure 6 summarizes the GQM approach we followed.

Table 6 presents the statements that we used to represent the questions in the questionnaire that participants answered.

Questions Q1.1–Q1.4 were used to characterize the portal usefulness, questions Q2.1–Q1.2 were used to collect data on the level of usability. Question Q3.1 was used to evaluate TKMP functional correctness. Table 7 shows how to interpret the results. The lines should be read as "IF <<expression>> THEN <<interpretation>>". For example, the interpretation of Question 3.1 (Q3.1) is "IF M1+M2 > M4+M5 THEN the users do not notice inconsistencies when using the TKMP", where M1, M2, M4, M5 are the responses given by the participants (metrics). It is important to notice that M1 and M2 (see Table 5) are answers that totally or partially disagree with the question. On the other hand, M4 and M5 are answers that totally or partially agree with the question.

In addition to the questions created using GQM's con-

Table 4. Defined Goals and Questions

G1: Evaluate TKMP usefulness	**Q1.1.** What is the perceived usefulness of TKMP regarding creating software testing knowledge items? **Q1.2.** What is the perceived usefulness of TKMP regarding searching for software testing knowledge items? **Q1.3.** What is the perceived usefulness of TKMP regarding reusing software testing knowledge items? **Q1.4.** What is the perceived global usefulness of TKMP?
G2: Evaluate TKMP usability	**Q2.1.** To what extent do users recognize that it is easy to learn to use TKMP? (learnability) **Q2.2.** To what extent do users recognize that TKMP is appropriate for their needs? (appropriateness recognizability)
G3: Functional Correctness	**Q3.1** Do users notice inconsistencies when using TKMP?

Table 5. Metrics used in the GQM

MG.Q.1	Number of participants who strongly disagree
MG.Q.2	Number of participants who disagreed
MG.Q.3	Number of participants who neither agree nor disagree
MG.Q.4	Number of participants who agree
MG.Q.5	Number of participants who strongly agree

Table 6. Statements used to refer to the questions

Q1.1	TKMP is useful to create software testing knowledge items.
Q1.2	TKMP is useful to search for software testing knowledge items.
Q1.3	TKMP is useful to reuse software testing knowledge items.
Q1.4	I would use or recommend the TKMP.
Q2.1	I learned to use the TKMP quickly.
Q2.2	I recognize TKMP as being suited to my tester needs.
Q3.1	I did not notice inconsistencies when using the TKMP.

ceptual level, at the end of the questionnaire, we present three open questions to professionals in order to allow the participant to externalize their opinion about the TKMP in terms of good points, bad points, and general comments.

(II) Data Collection. The data used to evaluate the TKMP were based on the metrics presented above. To collect the data, we requested experts in software organizations to use TKMP to perform activities to create, validate and search for knowledge items. After using the tool, 43 participants answered a questionnaire containing the questions previously presented.

Considering the participants' profile, out of these 43, 8 hold Doctoral degrees, 13 hold Masters, 22 finished

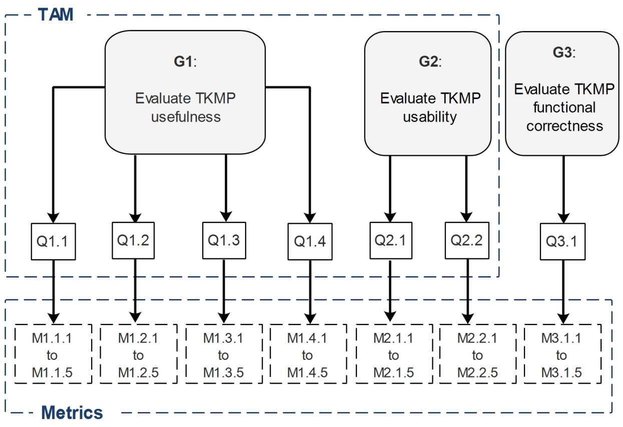

Figure 6. GQM approach to evaluate the TKMP

undergraduate programs. All of them are from the Software Engineering area and they have an average of six years of experience in the area. In relation to software testing knowledge, 42.9% of participants reported having basic knowledge, 37.2% reported having intermediate knowledge, and 23.3% considered having advanced knowledge on software testing. A summary of the responses given by the participants is shown in Table 8. This table shows the number of responses according to the goals, questions and metrics used.

(III) Interpretation. Figures 7, 9 and 10 present charts that show the answers per question used in our GQM model. These answers were interpreted according to Table 7:

Goal 1: Evaluate TKMP usefulness - Figure 7 presents the chart generated from the answers related to TKMP usefulness. Applying the interpretation expressions shown in Table 7, in relation to this goal, the results show that the participants considered TKMP a useful tool for managing software test knowledge items.
Regarding evaluating TKMP usefulness, we also carried out an analysis separating the 43 participants by professional position: professionals directly related to software development companies (23 professionals); and professionals directly related to scientific research (22 participants). This separation by position allowed us to infer how the software industry and the academic environment view the usefulness investigated topic. Figure 8 presents the chart generated from the answers related to TKMP usefulness by position. In general, analysis of the metrics for this chart, both for industry professionals and for researchers, TKMP is a type of tool that they would use or indicate, especially for research-related professionals (14 strongly agree).
Despite the interest, industry professionals presented a lower perception of the usefulness of the TKMP than academic researchers. In the SM conducted by Souza et al. (2015a) the main problems reported on the implementation of KM initiatives in software testing in the organization were investigated. The main problems mentioned were that KM systems are not yet appropriate; employees are normally reluctant to share their knowledge and increased workload. We believe that these problems may be related to the participants' responses.

Table 7. Results interpretation

01	For $Q1.i$, i=1 to 4: M1+M2 > M4+M5	TKMP is not useful for managing software testing knowledge items.
02	For $Q1.i$, i=1 to 4: M1+M2 < M4+M5	TKMP is useful for managing software testing knowledge items.
03	For $Q1.i$, i=1 to 4: M3 > M4+M5 or M1+M2 = M4+M5	We cannot say that TKMP is useful to manage software testing knowledge items
04	To $Q2.1$ and $Q2.2$: M1+M2 > M4+M5	TKMP cannot be easily used to manage software testing knowledge items.
05	To $Q2.1$ and $Q2.2$: M1+M2 < M4+M5	TKMP can be easily used to manage software testing knowledge items.
06	To $Q2.1$ and $Q2.2$: M3 > M4+M5 or M1+M2 = M4+M5	We cannot say that TKMP can be easily used to manage software testing knowledge items.
07	To $Q3.1$: M1+M2 > M4+M5	TKMP can be considered functionally correct.
08	To $Q3.1$: M1+M2 < M4+M5	TKMP cannot be considered functionally correct.
09	To $Q3.1$: M3 > M4+M5 or M1+M2 = M4+M5	We cannot say if TKMP is functionally correct or not.

Table 8. Results Summary

	Questions	Metrics					
Goal		M1	M2	M3	M4	M5	Total
G1	Q1.1	0	0	0	19	24	43
	Q1.2	2	2	8	17	14	43
	Q1.3	0	1	7	16	19	43
	Q1.4	0	4	9	8	22	43
G2	Q2.1	0	2	11	17	13	43
	Q2.2	1	0	11	17	14	43
G3	Q3.1	3	5	14	14	7	43

On the other hand, in the academic area, there is considerable growth in research in KM and Software Engineering. In 2008, Bjørnson and Dingsøyr (2008) already presented the growing interest in research on KM in software engineering. This growing interest continues to these days (Menolli et al., 2015; Vasanthapriyan et al., 2015; Pinto et al., 2018; Napoleão et al., 2021).

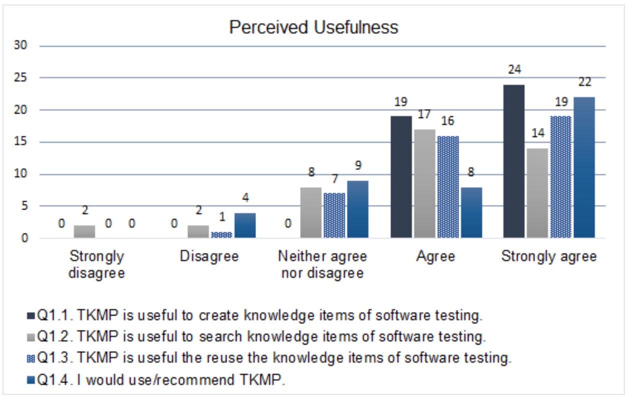

Figure 7. Questions and answers related to usefulness of TKMP

Goal 2: Evaluate the usability of TKMP - Figure 9 presents the chart generated from the answers related to usability. The results showed the participants considered that TKMP can be easily used to manage software testing knowledge items.

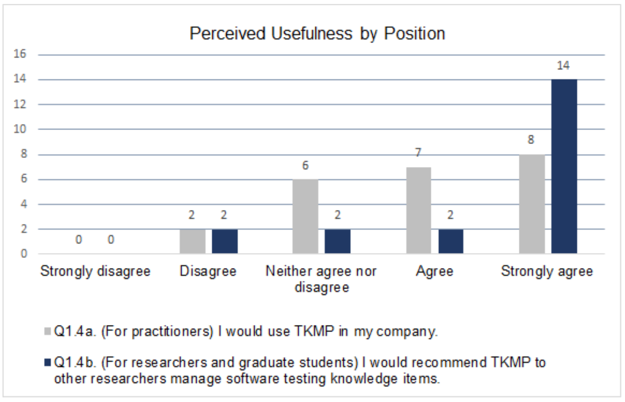

Figure 8. Questions and answers related to usefulness by position

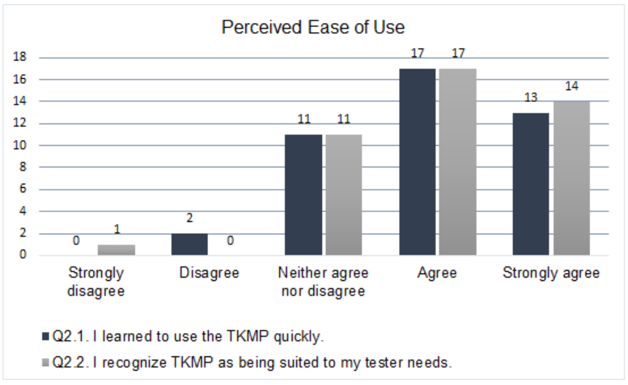

Figure 9. Questions and answers related to usability of TKMP

Goal 3: Evaluate the functional correctness of TKMP - Figure 10 presents the chart related to functional correctness. The results show that TKMP can be considered functionally correct. However, even the metrics pointing out that most of the participants did not find inconsistencies to the point that they were not able to use TKMP, by Figure 10, it is possible to notice that the participants found inconsistencies in TKMP. We consider this a normal result since TKMP is still a prototype.

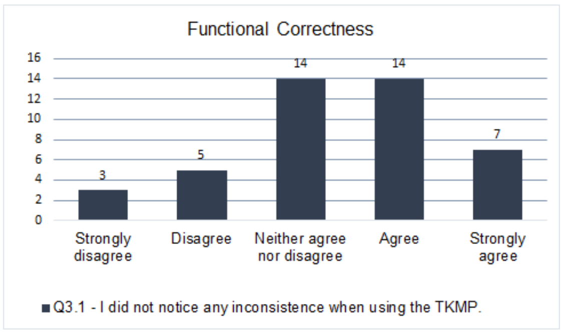

Figure 10. Question and answers related to functional correctness of TKMP.

As mentioned in the questionnaire planning, we present three open questions to professionals to externalize good points, bad points, and general comments about the TKMP. As can be seen in Figure 7 (usefulness), Figure 9 (ease of use) and Figure 10 (functional correctness), some professionals chose the option "Strong disagreement or Disagree" in the TKMP evaluation. We analyzed open-ended responses that practitioners wrote to identify improvements to the tool and consequently the approach.

When analyzing the responses of these 10 participants, most of the comments are related to functional correctness analysis (Figure 10). In general, we noticed that many of the observations were more related to the small inconsistencies identified in the tool. Two of the professionals, for example, mentioned that the search for knowledge items could be improved to be faster. One of the professionals mentioned that when there is a lot of data to be returned in a database, in the implementation it is possible to use more advanced strategies to optimize this process. Other comments for improvements were: the system would better work with images; allow access to an instructional help in any part of the tool; keep all fields in the tool as case sensitive; and allow sharing of information via email, as well as sending an email of evaluations of knowledge items performed. It is worth noting that the TKMP is a prototype considered as a proof of concept. Despite this, all suggestions for improvements will be considered in the evolution of this research.

It's also possible to notice, especially, by the charts of Figures 7 and 9, that a considerable number of participants chose the option "Neither agree nor disagree" for TKMP usefulness and usability. When analyzing the responses of these participants separately (15 participants), we did not find any pattern that justified this choice. We only noted that concerning the knowledge level in software testing, 10 participants mentioned having basic or intermediate knowledge. It is not possible to say, but we believe that a low time of knowledge about software testing may have some influence on the answer about TKMP utility and usability.

4.6 Other Partial Applications of OntoT-KM

We also started to apply OntoT-KM in software organizations. Three companies evaluated OntoT-KM and TKMP. First, we conducted the diagnostic and scope definition activities in these three companies by applying a questionnaire based on the survey presented in Souza et al. (2015b).

Respondents to the questionnaire were software testers responsible for the software testing activities within the companies. For privacy reasons, we do not mention the company's names. However, some characteristics are: located in Brazil; medium sized software organizations; the main products they develop are systems for the fiscal area, such as an electronic fiscal receipt, metrology, and also customized systems to meet the needs of customers from diverse segments. The main diagnosis results by the three companies are: (i) "Test Case Design" activity was the most useful; (ii) "Test Environment Structuring" was the testing planning activity in which KM is most useful; (iii) "Human resource" and "Software Resource" are considered the resources from which it is quite important to have the knowledge available at the time of setting the test environment; (iv) the explicit knowledge was considered more important than tacit knowledge; (v) "Test Plan" and "Test Case" were considered the artifacts most reusable ones; (vi) There is no formal instrument for KM within the three companies; and (vii) "Increasing the testing process efficiency" and "Best test case selection" are the main expected benefits of applying KM in software testing. The results were very close to the results obtained in the general survey applied to the 86 participants in Souza et al. (2015b). From the diagnosis results in the companies, it was possible to establish the scope for software testing initiatives. The *test plan* definition and *test case design* were considered the software testing activities to be first supported, and test cases the main knowledge item to be managed.

Until now, we have not conducted the remaining activities of the OntoT-KM approach (develop a KMS Testing, Load Existing Knowledge Items, and Evaluate the Testing KMS), although companies have shown interest in developing their own KMS solution. We intended that companies would also use TKMP. We proposed to the three companies the use of TKMP already developed from the research project's scope since the diagnosis results were similar. Companies were at ease in uploading the organization's data in TKMP or registering new data if they wish. The purpose was to analyze if an already existing KM tool, such as TKMP, could be customized by the organization to meet its current needs. Some suggested customizations were: (i) to implement a traceability matrix among test cases and lessons learned in order to assist the test coverage; (ii) to develop a repository of artifacts (historical basis); and (iii) turning TKMP into a plug-in to integrate with project management tools (e.g., JIRA, Redmine).

The two fronts analyzed in this study were well accepted by the three organizations that participated in the survey. The participants mentioned that it is interesting to have their solution for a KMS using OntoT-KM. However, they mentioned this would be possible if the company had a team to develop the system. On the other hand, it is also attractive to have a more general and open source KMS available to be customized by the company. We believe that the experience we had from evaluations performed both OntoT-KM, as well as TKMP, give us motivations and directions for future works, for example, we intend to consider some of the customization suggestions and enhancements to be implemented in TKMP since there is an intention to create a robust version of the portal to be made available to the Software Testing community.

4.7 Study Limitations

There were limitations in the study. The first limitation refers to the low representativeness of companies participating in the study (3 companies). Validating an approach, as OntoT-KM in a real environment, needs the authorization and trust of the organization to use its data and information, and allocate employees for system development. However, we noticed an enormous barrier to this. Several other companies were invited to apply OntoT-KM. While they recognize the benefits from KM in software testing, many refused to participate. The invited organizations mentioned that the idea of implementing KM in the organization, even with an existing tool, could generate an increased workload. This position is in line with the results detected through the SM conducted in Souza et al. (2015a). Shortage of time is also a potential risk to incorporate KM principles in software testing because knowledge sharing can imply increasing the employee workload and costs. We intend to continue inviting software companies to participate in the research and look for strategies that allow the company to feel safe in relation to use or build a KMS, for example, allow the company to install the system on-premises and on their database server.

A second limitation of this research concerns the sample size of the software engineering practitioners that answered the questionnaire. 43 practitioners answered the questionnaire. Of these, 23 are professionals directly related to software development companies. The results cannot be generalized. Therefore, we intend to replicate this survey as many software practitioners as possible in real projects in the industry. In addition, we also intend to conduct interviews with these professionals. The interview purpose is to better understand the responses that professionals have about TKMP, for example, to better understand the reason that led professionals to shore up the option "Neither agree nor disagree", as can be seen in the Figures 7, 9 and 10.

Another limitation is related to step 1 of OntoT-KM. This step was not employed on both projects (ICAMMH and SIA). For this step, we used the results of a survey. The diagnosis step was not made exclusively for the projects in question. However, some survey participants were team members and leaders of the ICAMMH and SIA projects. We believe that the participation of these team members may have helped to achieve a specific diagnosis for the projects under study.

5 Related Work

Different approaches to the development of KMSs can be found in the literature. Dehghani and Ramsin (2015) provided a review of seven methods for KMS development. In general, these methods provide activities, principles, and techniques intending to apply KM in the organizations (R-Montano et al., 2001; Calabrese and Orlando, 2006; Chalmeta and Grangel, 2008; Iglesias and Garijo, 2008; Sarnikar and Deokar, 2010; Moteleb et al., 2009; Amine and Ahmed-Nacer, 2011). Some of these KMS methodologies are presented below.

Chalmeta and Grangel (2008) presented a methodology called KM-IRIS. KM-IRIS was defined on a general level that can be used as a guide to managing knowledge in any kind of organization. The methodology is divided into five phases: (i) Analysis and Identification of the target knowledge; (ii) Extraction of the target knowledge; (iii) Classification and representation; (iv) Processing and storage. In this activity an operational KMS is implemented; and (v) Utilization and continuous improvement by using the KMS. Chalmeta and Grangel (2008) mention that ontologies can be used in the first phase of the methodology, that is, after identifying the knowledge, this knowledge can be detailed building on an ontological classification so that it can be represented, processed, and used in a later phase. Ontologies are also suggested by Chalmeta and Grangel (2008) to be used in the second phase of the methodology. OntoT-KM also has guidelines that identify target knowledge, called knowledge items, and this item should be ranked. OntoT-KM is based on a test ontology and the diagnostic phase of the test environment, as well as the development of KMS, are strongly related to this ontology.

In (R-Montano et al., 2001), a methodology to develop a KMS was presented. The phases of this methodology are as follows: (i) A strategic planning; (ii) Models logical and physical aspects by specifying the strengths and weaknesses of the organizational KM process; (iii) Development of the KMS prototype; (iv) verification and validation the KMS through practical usage of the system; and (v) Deploy and maintain the KMS. Similar to the methodology of (R-Montano et al., 2001), OntoT-KM also proposes a planning stage, called diagnosis, as well as generation of models to support the construction of a KMS and its validation. However, these main activities in OntoT-KM are supported by a software testing ontology.

Calabrese and Orlando (2006) presented a methodology that consists of 18 phases: (i) KM principles and governance; (ii) Organizational structure and sponsorship; (iii) Requirements analysis; (iv) Measurement; (v) Knowledge audit; (vi) Initiative scoping; (vii) Prioritization; (viii) Technology solution assessment; (ix) Planning the development of the KMS; (x) Knowledge elicitation; (xi) Building the KMS; (xii) Verifies and validates the KMS; (xiii) Review and update the KMS; (xiv) Knowledge maintenance processes; (xv) Communication and change management; (xvi) Train and publish the KMS; (xvii) Maintenance and support; and (xviii) Measurement and reporting. In general, the methodology presented by Calabrese and Orlando (2006) is the detailing of the process for constructing a KMS. For example, in the OntoT-KM process (Figure 2) it is possible to notice that after the evaluation activity of KMS testing, improvements can be the system returning to previous process activities. However, we do not treat this action as an explicit activity but in the form of a relationship arrow. On the other hand, in the case presented by Calabrese and Orlando (2006), this action is considered in phases (xiii), (xiv), (xv), (xvii) and (xviii).

In (Sarnikar and Deokar, 2010), a methodology is presented to direct the development process based on the workflows within the organization. The methodology consists of 7 different design steps: (i) Business process model development of the organization; (ii) Knowledge intensity identification; (iii) Requirements' identification; (iv) Knowledge sources identification; (v) Knowledge reuse assessment; (vi) Task-user knowledge profile development; and (vii) Designs the system components to support the tasks investigated in previous phases. Different from the OntoT-KM process and also from the other methodologies presented in (Dehghani and Ramsin, 2015), the (Sarnikar and Deokar, 2010) methodology presents the design and construction of KMS only in its last phase.

Iglesias and Garijo (2008) presented a methodology that is not specifically targeted at developing a KMS but can be effectively used for this purpose. Iglesias and Garijo (2008) proposed the methodology MASCommonKADS that extends object-oriented and knowledge engineering techniques for the conceptualization of multi-agent systems. The phases of the methodology are as follows: (i) Obtains the initial view of the problem domain; (ii) Discovers system requirements; (iii) Designs the system; (iv) Develops and tests; and (v) Operates and maintains the system. In the Designs phase, an initial set of agents are determined and a model is developed. The communication between the agents is expressed in an ontology.

In (Amine and Ahmed-Nacer, 2011), an ontology-based agile methodology is presented to develop a KMS to reduce the risks of component-based development through managing the knowledge needed for component selection, update, and maintenance. The phases are as follows (the last four phases are iterative): (i) Initialization. The main objective of this phase is to have the deepest understanding possible of the organization. In this phase the creation of an initial ontology of the organization domain can be conducted; (ii) Domain mapping. Continuously refines the problem domain ontologies created in the initialization phase; (iii) Profiles and policies identification; Specifies the authentication mechanisms and the level of system access allowed for each user; (iv) Implementation and personalization of the KMS; and (v) Verification and validation of the KMS. The phases of the methodology proposed by Amine and Ahmed-Nacer (2011) are very similar to OntoT-KM. As with Amine and Ahmed-Nacer (2011), we also use the resources of an ontology. However, unlike Amine and Ahmed-Nacer (2011), the ontology we use is not created based on the organization but rather on an already validated domain ontology and that aims at establishing a common conceptualization about the software testing domain.

Finally, the methodology presented by Moteleb et al. (2009) aims at using practical experiences for developing KMSs in small organizations. The methodology is divided into five phases: (i) Sense-making that aims at investigating whether KMS development is a conceivable solution for the organizational problems; (ii) Categorize the conceivable solutions through communicating with the stakeholders; (iii) The system is designed based on the solutions presented in the previous phase; (iv) Specifies the appropriate technologies based on the technical, social and organizational features of the KMS; and (v) Monitors and maintains the KMS. OntoT-KM also analyzes the solution for the organization in the diagnostic phase, as well as the design to construct the KMS. However, as mentioned, OntoT-KM is supported by software testing ontology, since this domain is the goal of OntoT-KM.

Table 9 presents a brief comparison of related work, discussed above, that presented approaches to the development of a system for supporting KM.

To the best of our knowledge, there is no method devoted to developing a KMS for supporting KM in software testing. In this way, we compared the system developed using OntoT-KM (TKMP) with other works addressing KM in software testing. These works are some of the ones selected in the mapping study on initiatives applying KM in software testing presented in (Souza et al., 2015a). Thus, the studies retrieved in this mapping were used here as a baseline for comparison with our work. Most of the studies providing automated support for managing testing knowledge employing a KMS. In addition, the mapping results point out that test case reuse has been the major focus of these initiatives. These results are in line with the findings of the survey that guided us in the development of TKMP, concerning the fact that test cases are the main knowledge item to be managed.

In (Janjic and Atkinson, 2013), an automated test recommendation approach that proactively makes test case suggestions while a tester is designing test cases was presented. They developed a prototype of an automated, non-intrusive test recommendation system called Test Tenderer. A search engine, called SENTRE, uses the current test case to perform a search for reusable, semantically matching components. Analogously to (Janjic and Atkinson, 2013), test case design was considered the software testing activity to be supported by TKMP. However, Test Tenderer addresses unit testing, while TKMP is more general. Although Janjic and Atkinson say that SENTRE performs a search for reusable, semantically matching components, the heuristics applied are name-based searches. In TKMP, in turn, the knowledge repository is structured based on ROoST, which is also used as the basis for the search functionality. Finally, Test Tenderer works non-intrusively in the background and smoothly integrates into normal working environments. Thus, the developer's normal working practices are not disturbed, and they only need to break away from the task of writing new test cases to consider already existing tests suggested by the recommendation engine. TKMP, on the other hand, does not proactively suggest test cases. Testers must make a query for retrieving similar test cases.

The technologies to support KM in software testing were another important question investigated by the mapping. The mapping showed that Knowledge Maps/Yellow Pages seem to have good results. A Knowledge Map contains information about experiences that employees possess. In (Liu et al., 2009), for instance, a KM model whose one of its main components is a Knowledge Map repository was created. The system identifies, utilizing statistics, the staff with some knowledge, improving the culture of knowledge-sharing in the enterprise. Analogously, TKMP also provides a Yellow Page

Table 9. Characteristics for different approaches to the development of KMSs

Approach	Objective	Number Phases	Ontology	Evaluation
Chalmeta and Grangel (2008)	Methodology for directing the process of developing and implementing a KMS in any type of organisation	5	Ontologies are suggested to be used in the steps (i) Analysis and Identification of the target knowledge and (ii) Extraction of the target knowledge	The methodology was applied to a large textile enterprise
R-Montano et al. (2001)	Recommendations to develop a KMS	5	-	-
Calabrese and Orlando (2006)	Process for a comprehensive KMS	12	-	A sensitivity/realism assessment using an actual configuration management application to demonstrate the utility of the process was conducted
Sarnikar and Deokar (2010)	A design process for KMS	7	-	The design process was validated by demonstrating the feasibility of the proposed design process and comparing the approach with others modeling approaches
Iglesias and Garijo (2008)	Methodology MASCommonKADS that extends object-oriented and knowledge engineering techniques for the conceptualization of multi-agent systems	5	An ontology can be used in the communication between the agents	A case study was conducted in a travel agency context
Amine and Ahmed-Nacer (2011)	Implementation of KMS using Component-based software engineering (CBSE)	5	An ontology-based agile methodology was used	A case study of the application of the methodology was conducted in a software organization
Moteleb et al. (2009)	Use of practical experiences for developing KMSs in small organizations	5	-	The approach was validated in practice by an inquiry into a number of problems experienced by particular organizations
OntoT-KM	Development of an ontology-based approach for KM in Software Testing	5	An a Reference Ontology on Software Testing was used	A KMS was development as proof of concept. KMS was evaluated in terms of usefulness, usability, and functional correctness

feature. Li and Zhang (2012) presents a Knowledge Management Model and one of the elements of this model is also a knowledge map. This model is based on an ontology of reusable test cases. However, this ontology has limited coverage when compared with the ROoST.

6 Conclusions

This work presents our experiences in developing an approach to assist in launching KM initiatives in software testing. OntoT-KM provides guidelines to apply KM with the development of KMSs and based on a software testing ontology. Although there are approaches for developing KMSs Dehghani and Ramsin (2015), to the best of our knowledge, there is no approach devoted to developing a KMS for supporting KM in software testing. In this respect, OntoT-KM is an original contribution. Results show that the developed KMS from OntoT-KM is a potential system for managing knowledge in software testing, so, the approach, can guide KM initiatives in software testing.

An approach like OntoT-KM can support different scenarios in software development companies. Organizations that develop different products or product lines, for example, have a large turnover of knowledge when compared to organizations that build specific software for each client/project (Matturro and Silva, 2005). Hence, the reuse of testing knowledge becomes more frequent in the later phases of software development. Thus, a KM system, such as TKMP, would allow searching for solutions to similar problems registered in the tool. Reuse is related not only to similar test cases, but also to lessons learned, best practices, and patterns of behavior in the project that can be identified by item mined and that can be reused or at least assist in project decisions.

In relation to OntoT-KM evaluation, in this work, we intended to evaluate the approach, as well as the generated KMS. Now we intend to apply the diagnosis to as many software development companies as possible to reach a common scope in order to be developed in a general KMS. This KMS will be part of an environment already maintained by this research project, called Software Engineering KNOWledge management diagnosis (SEKNOW) Santos et al. (2019). Currently, SEKNOW was developed only to analyze KM in software development organizations (diagnostics step), however, given the evolution of research, SEKNOW has been undergoing adaptations to meet more activities related to KM and software organizations. We also intend as future work to extend TKMP considering other conceptualizations established by ROoST. We also intend to conduct more experimental studies to confirm the results of the evaluations discussed in this paper.

As mentioned earlier, we will apply KM diagnosis in software development companies that maintain different project domains with agile or traditional developments. The objective of KM diagnosis is to measure the organization's current state of KM. KM diagnosis can help the company to understand the real needs before devoting costly efforts to KM implementation and thus better target KM application initiatives at strategic points (Bukowitz and Williams, 2000). Conducting KM diagnostics in different domains of software development can shows how KM activities are present in environments with agile or traditional practices. For this reason, we have been conducting the synthesis on KM and Agile Software Development (ASD) (Napoleão et al., 2021), and it will certainly be considered in the next stages of this project. Just like ADS, Purpose Development and Operations (DevOps) practices are also strongly related to KM. DevOps is a methodology that combines flexibility with rigorous testing and communication routines, aiming to deliver software efficiently and quickly (Mishra and Otaiwi, 2020). The adoption of DevOps in an organization provides many benefits including quality but also brings challenges to an organization, for example, knowledge reuse. It is in our interest to use the study conducted in this work in an organization that adopts DevOps and measures how much is possible to manage knowledge in software testing in this context.

Acknowledgements

The first author would like to thank Professor Ricardo de Almeida Falbo (in memoriam) for successfully leading this work and sharing your valuable advice. The authors would like to thank: Brazilian Aeronautics Institute of Technology (ITA) and the Brazilian Agency of Research and Projects Financing (FINEP) - Project 5206/06-ICAMMH; and the SIA Project for providing the data. Brazilian funding agency CNPq Project 432247/2018-1. All participants that used the TKMP and answered the evaluation questionnaire are also duly acknowledged.

References

Abran, A., Bourque, P., Dupuis, J., and Moore, W. (2004). Guide to the software engineering body of knowledge - SWEBOK. Technical report, A project of the IEEE Computer Society Professional Practices Committee.

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases*, pages 487–499.

Amine, M. and Ahmed-Nacer, M. (2011). An agile methodology for implementing knowledge management systems: a case study in component-based software engineering. *Software Engineering Applications*, 5:159–170.

Andrade, J., Ares, J., Martinez, M., Pazos, J., Rodriguez, S., Romera, J., and Suarez., S. (2013). An architectural model for software testing lesson learned systems. *An architectural model for software testing lesson learned systems*, 55:18–34.

Basili, V. and Rombach, H. D. (1991). Support for comprehensive reuse. *Software Engineering Journal*, 6:303–316.

Basili, V. R., Caldiera, C., and Rombach, H. (1994). Guide to the software engineering body of knowledge - SWEBOK. Technical report, Goal Question Metric Paradigm, New York: John Wiley & Sons.

Bjørnson, F. O. and Dingsøyr, T. (2008). Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50:1055–1068.

Black, R. and Mitchell, J. L. (2011). *Advanced software testing*. Rocky Nook, USA, 3 edition.

Bukowitz, W. and Williams, R. L. (2000). *The knowledge management fieldbook*. Financial Times Prentice Hall, Great Britain.

Burnstein, I. (2003). *Practical Software Testing: A Process-oriented Approach*. Springer Professional Computing, New York, 3 edition.

Calabrese, F. and Orlando, C. (2006). Deriving a 12-step process to create and implement a comprehensive knowledge management system. *Journal of Information and Knowledge Management Systems*, 3(36):238–254.

Chalmeta, R. and Grangel, R. (2008). Methodology for the implementation of knowledge management systems. *Journal of the American Society for Information Science and Technology*, 5(59):742–755.

Davenport, T. H. and Prusak, L. (2000). *Working knowledge*. Harward Business School Press, Boston, USA, 2 edition.

Davis, F. D. (1993). User acceptance of information technology: system characteristics, user perceptions and behavioral impacts. *International Jounal of Man-Machine Studies*, 38:475–487.

Dehghani, R. and Ramsin, R. (2015). Methodologies for developing knowledge management systems: an evaluation framework. *Journal of Knowledge Management*, 19(4):682–710.

Falbo, R. A. (2014). Sabio: Systematic approach for building ontologies. In *8th Intern. Conference on Formal Ontology in Information Systems*.

Falbo, R. A., Arantes, D. O., and Natali, A. C. C. (2004).

Integrating knowledge management and groupware in a software development environment. In *International Conference on Practical Aspects of Knowledge Management*, pages 94–105.

Falbo, R. A., Barcellos, M., Nardi, J., and Guizzardi, G. (2013). Organizing ontology design patterns as ontology pattern languages. In *Extended Semantic Web Conference*, Montpellier.

Falbo, R. A., Ruy, F. B., Guizzardi, G., Barcellos, M. P., and Almeida, J. P. A. (2014). Towards an enterprise ontology pattern language. In *Symposium On Applied Computing*, Gyeongju.

Fayyad, U., Gregory, P., and P.Smyth, P. (1996). From data mining to knowledge discovery in databases. *American Association for Artificial Intelligence*, pages 37–54.

Fischer, G. and Ostwald, J. (2001). Knowledge management: problems, promises, realities, and challenges. *IEEE Intelligent Systems*, 16:60–72.

Herrera, R. J. G. and Martin-B, M. J. (2015). A novel process-based KMS success framework empowered by ontology learning technology. *Engineering Applications of Artificial Intelligence*, 45:295–312.

Iglesias, C. and Garijo, M. (2008). The agent-oriented methodology MAS-CommonKADS. In *Intelligent Information Technologies: Concepts, Methodologies, Tools, and Applications, Information Science*, pages 445–468.

ISO/IEC (2011). ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation(SQuaRE)Syste - m and software quality models.

Janjic, W. and Atkinson, C. (2013). Utilizing software reuse experience for automated test recommendation. In *International Workshop on Automation of Software Test*, pages 100–106, San Francisco.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical Report EBSE 2007-001, Keele University and Durham University, UK.

Li, X. and Zhang, W. (2012). Ontology-based testing platform for reusing. In *Intern. Conference on Internet Platform for Reusing*, pages 86–89, Henan, China.

Liu, Y., Wu, J., Liu, X., and Gu, G. (2009). Investigation of knowledge management methods in software testing process. In *Inter. Conference on Information Technology and Computer Science*, pages 90–94, Kiev.

Mathur, A. P. (2012). *Foundations of software testing*. Pearson Education in South Asia, India, 5 edition.

Matturro, G. and Silva, A. (2005). A knowledge-based perspective for preparing the transition to a software product line approach. In *International Conference on Software Product Lines*, pages 96–101, Berlin, Heidelberg.

Menolli, A., Cunha, M. A., Reinehr, S., and Malucelli, A. (2015). "old" theories, "new" technologies: Understanding knowledge sharing and learning in brazilian software development companies. *Information and Software Technology*, 58:289–303.

Mishra, A. and Otaiwi, Z. (2020). Devops and software quality: A systematic mapping. *Computer Science Review*, 38:100308.

Moteleb, A., Woodman, M., and Critten, P. (2009). Towards a practical guide for developing knowledge management systems in small organizations. In *European Conference on Knowledge Management*, pages 559–570.

Myers, G. J. (2004). *The art of software testing*. John Wiley and Sons, Canada, 2 edition.

Napoleão, B. M., Souza, E. F., Ruiz, G. A., Felizardo, K. R., Meinerz, G. V., and Vijaykumar, N. L. (2021). Synthesizing researches on knowledge management and agile software development using the meta-ethnography method. *Journal of Systems and Software*, 178:110973.

Nonaka, I. and Krogh, G. (2009). Tacit knowledge and knowledge conversion: controversy and advancement in organizational knowledge creation theory. *Organization Science*, 30:635–652.

Nonaka, I. and Takeuchi, H. (1997). *The knowledge-creating company*. Oxford University Press, Oxford, USA.

O'Leary, D. and Studer, R. (2001). Knowledge management: an interdisciplinary approach. *IEEE Intelligent Systems*, 16(1).

O'Leary, D. E. (1998). Enterprise knowledge management. *IEEE Computer Magazine*, pages 54–61.

Park, R., Goethert, W., and Florac, W. (1997). *Goal-Driven Software Measurement*. Handbook CMU/SEI-96-HB-002.

Pinto, D., Oliveira, M., Bortolozzi, F., Matta, N., and Tenório, N. (2018). Investigating knowledge management in the software industry: The proof of concept's findings of a questionnaire addressed to small and medium-sized companies. In *10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KMIS*, pages 73–82.

R-Montano, B., Liebowitz, J., Buchwalter, J., McCaw, D., Newman, B., and Rebeck, K. (2001). A systems thinking framework for knowledge management. *Decision Support Systems*, 31:5–16.

Rokunuzzaman, M. and Choudhury, K. P. (2011). Economics of software reuse and market positioning for customized software solutions. *Journal of Software*, 6:31–1029.

Ruy, F. B., Falbo, R., Barcellos, M., Costa, S. D., and Guizzardi, G. (2016). SEON: A Software Engineering Ontology Network. In *20th Inter. Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 527–542.

Santos, V., Salgado, J. G., Souza, E. F., Felizardp, K. R., and Vijaykumar, N. L. (2019). A tool for automation of knowledge management diagnostics in software development companies. In *Brazilian Conference on Software: Theory and Practice (CBSoft) - Tools Session*.

Sarnikar, S. and Deokar, A. (2010). Knowledge management systems for knowledge-intensive processes: design approach and an illustrative example. In *International Conference on System Sciences*, pages 1–10.

Souza, E. F. (2014). *Knowledge management applied to software testing: an ontology based framework*. Thesis in computer science, National Institute for Space Research (INPE), Brazil.

Souza, E. F., Falbo, R. A., Specimille, M. S., Coelho, A. G. N., Vijaykumar, N. L., Felizardo, K. R., and Meinerz, G. V. (2020). Experience report on developing an ontology-based approach for knowledge management in software testing. In *19th Brazilian Symposium on Software Quality - Experience Reports (SBQS '20)*, pages 1–10.

Souza, E. F., Falbo, R. A., and Vijaykumar, N. (2017). ROoST:Reference Ontology on Software Testing. *Applied Ontology*, 12:59–90.

Souza, E. F., Falbo, R. A., and Vijaykumar, N. L. (2013). Ontology in Software Testing: a Systematic Literature Review. In *Research Seminar Ontology of Brazil (ONTOBRAS)*, pages 71–82, Belo Horizonte.

Souza, E. F., Falbo, R. A., and Vijaykumar, N. L. (2015a). Knowledge management initiatives in software testing: A mapping study. *Information and Software Technology*, 57:378–391.

Souza, E. F., Falbo, R. A., and Vijaykumar, N. L. (2015b). Using lessons learned from mapping study to conduct a research project on knowledge management in software testing. In *41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 208–215, Madeira, Portugual.

Staab, S., Studer, R., Schurr, H. P., and Sure, Y. (2001). Knowledge processes and ontologies. *Intelligent Systems*, 16:26–34.

Storey, J. and Barnett, E. (2000). Knowledge management initiatives:learning from failure. *Journal of Knowledge Management*, 4:145–156.

Thrane, C. (2011). *Quantitative models and analysis for reactive systems*. Thesis in applied computing, Department of Computer Science - Aalborg University, Denmark.

Vasanthapriyan, S., Tian, J., and Xiang, J. (2015). A survey on knowledge management in software engineering. In *International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 237–244, Vancouver, BC, Canada.

Werner, J. (2014). *Reuse-Based Test Recommendation in Software Engineering*. PhD thesis, Universität Mannheiml, Mannheim. Zugl. als Druckausg. im Verl. Dr. Hut, München erschienen.

Witten, I. H., Frank, E., and Hall, M. A. (2005). *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 3 edition.

Yun, H., Ha, D., Hwang, B., and Ryu, K. (2003). Mining association rules on significant rare data using relative support. *Journal of Systems and Software*, 67:181–191.

Zack, M. and Serino, M. (2000). Knowledge management and collaboration technologies. In *Knowledge, Groupware and the Internet*, pages 303–315, Butterworth.