e-Informatica Software Engineering Journal, Volume 7, Issue 1, 2013, pages: 25–33, DOI 10.5277/e-Inf130103

A Knowledge-Based Perspective for Software Process Modeling

Noureddine Kerzazi*, Mathieu Lavallée*, Pierre-N. Robillard*

**Laboratoire de Recherche en Génie Logiciel, 2500 chemin de Polytechnique, Montr[Pleaseinsert\PrerenderUnicode{Âľ}intopreamble]al H3T 1J4, Canada, École Polytechnique de Montréal*

Noureddine.Kerzazi@Polymtl.ca, Mathieu.Lavallee@Polymtl.ca, Pierre.Robillard@Polymtl.ca

Abstract

As the acquisition and sharing of knowledge form the backbone of the software development process, it is important to identify knowledge discrepancies between the process elements. Explicit representation of the knowledge components within a software process model can provide a means to expose these discrepancies. This paper presents an extension of the Software and System Process Engineering Metamodel (SPEM), to be used as a new knowledge modeling layer. The approach, which is based on ontologies for knowledge representation, constitutes an explicit method for representing knowledge within process models. A concept matching indicator shows the state of the process model in terms of the concept gaps for each task within the process. This indicator could lead to more informed decision making and better management of the associated risks, in terms of team competency, documentation quality, and the training required to mitigate them.

1. Introduction

Software engineering is knowledge-intensive [1], and so the acquisition and sharing of knowledge form the conceptual backbone of software process modeling. However, the latest version of SPEM [2], the Object Management Group (OMG)'s de facto standard devoted to software process modeling, does not support this knowledge concern. We tackle the issue in this paper by presenting an extended SPEM framework which focuses on the knowledge-oriented perspective of process modeling. Our approach responds to the need to provide process engineers with the means to perform knowledge assessments within processes, and we do this by integrating knowledge concepts within SPEM-based software process modeling. This knowledge-oriented perspective has been integrated into our previous work related to the implementation of a domain-specific language for software process modeling (DSL4SPM tool) [3].

In addition, the paper tackles the issue of knowledge discrepancies in the software process model, with the purpose of providing an indicator of a knowledge gap relating to any activity represented in that model.

The benefit of introducing knowledge concepts into software process modeling is to enable the identification of knowledge gaps between prescribed activities and the available resources. When such a gap is identified, the project manager can evaluate the risks it represents, and provide the developers with additional training as required [4]. The objective of this SPEM extension is thus to provide process engineers with a mean to document domain knowledge constraints directly into their process models.

The paper is organized as follows: Section 2, Related Works, presents theoretical background related to ontologies and knowledge representation. Section 3, Extension of SPEM, describes the SPEM extension developed to enable knowledge integration and gap measurement. Section

4 presents the visualization of the knowledge gap measure, along with its interpretations. Finally, section 5 presents concluding remarks and suggestions for future work.

2. Related Works

The background theory of this work relates to three main areas: software process modeling, knowledge management theories, and the use of ontologies. Knowledge management has been extensively studied, in the social sciences literature in particular. The purpose of this section is to highlight the salient concepts of the theories on which this approach is based.

2.1. Knowledge Management Theories

According to Davenport and Prusak [5], Knowledge Management (KM) can be defined as a process that relates to three issues: knowledge creation, knowledge representation, and knowledge sharing. This subsection highlights representative theories from the fields of management and the cognitive sciences, management theories emphasizing knowledge creation and representation, and cognitive theories emphasizing knowledge representation and storage.

In considering the management point of view, Nonaka & Takeuchi [6] identify two types of knowledge: tacit (T) and explicit (E). Tacit knowledge is personal and context-specific. It cannot, by definition, be explicitly expressed by an individual, but it frames his behavior. In contrast, explicit knowledge is externalized knowledge, and it supports communication, either formally or informally, and does so independently of who "knows". Within a software process, tacit knowledge resides in roles, while explicit knowledge resides in artifacts.

In considering the cognitive point of view, Novak and Canas [7] define knowledge as a structured set of interrelated concepts. They argue that learning involves the assimilation (i.e. internalization) of new concepts into existing cognitive structures. This theory is also embraced by cognitive psychologists, who assert that we learn by assimilating new concepts and propositions into existing cognitive structures, rather than through rote memorization [8]. So, representing knowledge in a structure of concepts [9] is in agreement with how cognitive psychologists believe we store knowledge in the brain.

Moreover, concepts can be organized in an ontology to simplify manipulation, sharing and reuse. An ontology is defined as "a formal explicit description of concepts in a domain" [10].

According to Anderson [11], there are two kinds of knowledge: declarative knowledge and procedural knowledge. Declarative knowledge is the factual or conceptual knowledge that a person has. It can be described and shared with others (e.g. facts about a programming language). Procedural knowledge is the knowledge of how to perform tasks, and focuses more on action than on information. This second type of knowledge is difficult to describe, but is nevertheless important, particularly in problem-solving (e.g. the experience of using a debugger). Robillard [1] presents an integrated view of various knowledge concepts in software engineering. Even though there is an abundance of theoretical knowledge models in the literature, there are few methods for integrating knowledge into the software process modeling field, as reported by Bjørnson & Dingsøyr in their systematic review [12]. With the recognition that knowledge is the primary source of an organization's innovation potential, our aim in this work is to superpose the knowledge-oriented perspective over the activity-oriented layer of the software process model, in order to provide a method for representing and managing knowledge in the software development process. The first step is to extend the SPEM-based processes with attributes related to knowledge. In this way, cognitive theory is used to represent:

– the knowledge required to carry out a task (considered as the core of the action),
– the knowledge provided by the artifacts and roles linked to this task.

The second step is to compare the knowledge required for the SPEM elements, such as artifacts, roles, and tasks, in order to identify gaps. The final step is to visualize the knowledge matching between tasks.

2.2. Ontology for Knowledge Representation

We chose an ontological representation among other formalisms for three reasons. First, cognitive psychologists, like Ausubel [8], affirm that people do not learn by rote memorization, but rather by summarizing, structuring, and relating concepts, and then assimilating them into existing cognitive structures. Based on this theory, the ontology domain was developed with the main aim of building knowledge repositories using a tree of interrelated concepts [13]. Second, we want to represent knowledge bases in a modular way, so that the model would be extendable and scalable to accommodate new items in the future. Third, we need a standard representation (e.g. RDF/XML or OWL 2.0) to be able to share knowledge bases and reuse them in different contexts.

Recently, more and more researchers have been using ontologies to understand software processes [14, 15]. According to Anquetil et al. [14], software system maintenance is a knowledge-intensive task, which can be supported with an ontology, and such an ontology can provide a good understanding of the application. The authors argue that many of the difficulties associated with software maintenance originate from knowledge management problems, and they propose a technique for knowledge extraction. Ferreira et al. [15] propose the integration of ontologies into the model driven architecture (MDA) paradigm.

In this work, we implement an interface, in addition to the basic process modeling one, to import multiple external ontologies.

3. Extension of SPEM

SPEM 2.0 is the OMG's standard for software process modeling [2]. It is based on UML and is dedicated to describing the components of software processes. According to its specification, a software process is a set of activities, each of which is composed of one or more tasks performed by an abstract active entity, called a Role. The Role is responsible for one or more tangible entities, called Work Products. Roles describe the responsibilities and a set of skills to facilitate their assignment to real people. Work Products are pieces of information produced by, or used by, Tasks.

However, SPEM supports activity-oriented modeling, which focuses on a structural breakdown of activities, and not knowledge modeling. There is therefore a need to extend SPEM to take into account a knowledge-oriented modeling perspective.

3.1. Integrating Knowledge Component into SPEM

Figure 1 shows the conceptual integration of the knowledge component into SPEM, the gray classes coming from the SPEM 2.0 specification. Note that *Work Product Use* is the generic term for the inputs and outputs of tasks, such as Artifacts, in the specification. The concept-based knowledge repository is structured as a tree of the *ConceptNode* element.

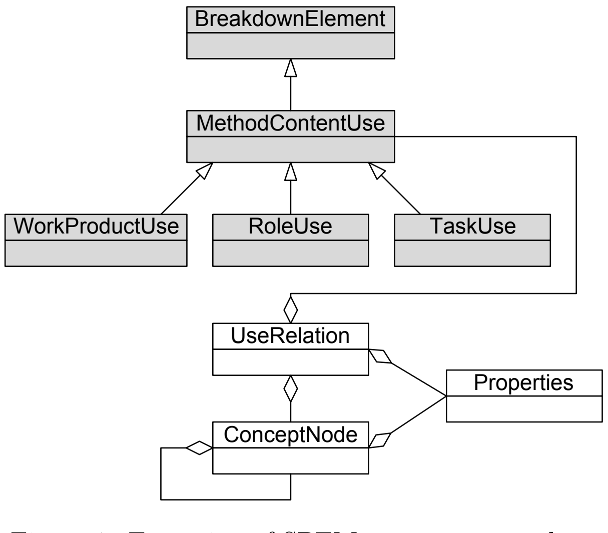

Figure 1. Extension of SPEM to support ontology. Elements in grey derive from the SPEM specifications. The extension is in white

The *ConceptNode* class represents a concept of the repository's tree-structured ontology. A node is associated with a list of properties. The *ConceptNode* is linked to process elements through the Relationship class. The Relationship class has one property, which enables the mod-

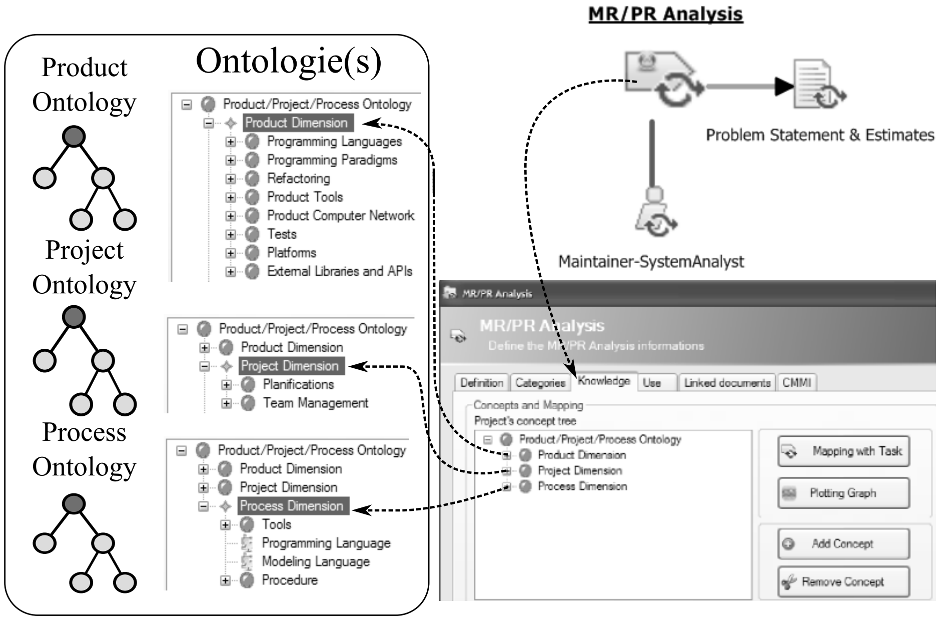

Figure 2. The specific form used to indicate the knowledge concepts needed for a task

eler to assign concepts to the process elements in a declarative way, a procedural way, or both. In this model, concepts do not yet have properties. The only relationship possible between concepts is an inheritance relationship,"*is-a*" . Future extensions may permit the expression of more properties and relationships.

According to Nosek and Roth [16], a visual representation of knowledge is clearly better for comprehension and conceptualization than a textual one. This paper presents visual modeling instead of the textual descriptions common in a number of specialized knowledge tools.

Figure 2 shows the specific form that allows knowledge modeling. The task is the core element of the process model; the process engineer has to indicate which knowledge concepts are needed to carry out each task, and, for each concept, the relationship with the process element (e.g. procedural and/or declarative) must be specified. The process engineer retrieves the related concepts required for each task from the concept ontology.

As shown in the upper-left corner of Figure 2, the ontology is organized according to three dimensions: product, project, and process. This multi-dimensional ontology approach (i.e. merging of ontologies) was motivated by validation experiments. The feedback of the participants revealed the need for a different level of knowledge. Some participants wanted a process ontology (e.g. role skills, artifacts, and guidance). Others asked for a project management ontology (e.g. assignment of human resources to roles, competency management). Still others were interested in a product ontology (e.g. API documentation, programming languages, techniques for understanding the source code, etc.). The results showed the need for a flexible and articulated tool, which can accommodate ontologies for multiple levels of knowledge.

3.2. Concept Maps for Knowledge Representation

The following four steps are required to evaluate the concept mapping between SPEM elements, which refers to process entities such as Activities, Roles, and Artifacts, as defined in the SPEM specification.

1. **Parameterization.** The first step is to specify the ontology, which is the formalization of the concept structure. The process engineer first loads a default or adapted concept tree. Then, the project team gathers and organizes the knowledge concepts relevant to the project context. As seen in Figure 2,

three concept trees are proposed by default, each one linked to an abstract level of knowledge (process, project, and product). Each concept may have a finer-grained structure represented with a number of attributes.

2. **Modeling.** For every task in the process model, references are set to a subset of concepts required for achieving the task. For every concept, the process engineer specifies the way in which the concept is required, following Anderson's model: declarative and/or procedural. For all incoming links to each task (e.g. from Work Products, Roles, Guidance), references are set to a subset of concepts provided by this element.
3. **Compute mismatch.** For every task, the system searches all incoming links, retrieves the concepts provided, and compiles the results. A concept required by the task is considered fully mapped if it can be provided by at least one of the input elements linked to this task. It is considered inadequately mapped if it is partially mapped (e.g. provided as declarative, when it is required to be procedural). Finally, it is considered not mapped if it is not provided by any SPEM element linked to this task.
4. **Visualize the results.** The concept matching indicator displays the results of concept mapping (as shown in Figure 3 below, which depicts knowledge mismatches). The user can visualize the resulting mismatches between prescribed knowledge and the available resources.

3.3. Algorithm for Knowledge Mapping

A formalism based on the conventional Vector Space Model (VSM) [17] is used to enable a simple visualization of the knowledge mismatch observed. VSM is used to compute the similarity between the knowledge required to carry out each of the prescribed tasks and the knowledge provided by the available roles, artifacts, and guidance. Cosine similarity measures are used to compute the similarity between two vectors of concepts characterized by n attributes (n-dimensional space). The results of this mapping are used to build the concept matching indicator. The first step is to link the process model to the ontology file, which contains a set of n concepts (Cn) relevant to the whole process. Thus, the knowledge required to carry out a given task ti is represented by a vector:

$$\overrightarrow{t_i} = \{C_1, C_2, ..., C_n\} * \overrightarrow{I} \quad (1)$$

where $I_i = 1$ if the concept is required, and 0 if not. According to Anderson [10], concepts can be classified in two categories: procedural or declarative, both of which are denoted by a vector (p,d). To simplify the illustration, we present only the Anderson attribute for the knowledge concept, which will be sufficient for most practical purposes. Let t'_T be the vector representing a set of typed concepts required for task T:

$$\overrightarrow{t'_T} = \{C_1(p,d)_{t'}, C_2(p,d)_{t'}, ..., C_n(p,d)_{t'}\} * \overrightarrow{I} \quad (2)$$

The next step is to link the source elements (i.e. other linked SPEM elements, such as Role, Artifact, and Guidance) of each task. These elements gather the concepts provided, instead of the concepts required. Let p'_E be the vector representing a set of typed concepts provided by a given element E:

$$\overrightarrow{p'_E} = \{C_1(p,d)_{p'}, C_2(p,d)_{p'}, ..., C_n(p,d)_{p'}\} * \overrightarrow{I} \quad (3)$$

The similarity between the vector t'_T and the p'_E of each of the elements provided is obtained by the cosine calculation. This gives a way of interpreting the quality of the mapping between the set of required knowledge concepts for each Task and the set of knowledge concepts provided.

$$\| p'_E \| = \sqrt{C_1(p,d)^2_{p'} + ... + C_n(p,d)^2_{p'}}$$

$$\| t'_T \| = \sqrt{C_1(p,d)^2_{t'} + ... + C_n(p,d)^2_{t'}}$$

$$D = \sqrt{\sum_{i=1}^{n} (C_i(p,d)_{t'} - C_i(p,d)_{p'})^2} \quad (4)$$

$$\cos\theta = \frac{\| p'_E \|^2 + \| t'_T \|^2 - D^2}{2 \| p'_E \| \| t'_T \|}$$

where the cosine values vary between 0 and 1:

p' values	t' values	Deviation
C1(p)+C2(d)+C3(p,d)	C1(p)+C2(d)+C3(d)	30°
C1(p,d)+C2(d)+C3(d)	C1(p,d)+C2(d)+C3(d)	0°
C1(p,d)+C2(p)+C3(p)	C1(p)+C2(p,d)+C3(d)	60°

Table 1. Calculation example

– If the required and provided concept vectors coincide completely, which means a perfect, complete concept mapping, the cosine is equal to 0.
– If the two vectors have no concepts in common, it means that all the concepts are unmapped and/or unused, and the cosine is equal to 1.
– Otherwise, the result mapping should be between 0 and 1.

The cosine angle values range between 0° and 90°. To improve readability all angle values are doubled, which enables a two quadrants (180°) representation, as shown in Figure 3.

Table 1 provides some examples of the calculation method.

4. Concept Matching Visualization Examples

As illustrated in Figure 3, the graph is divided into three zones (dark grey, medium grey, and light grey). An arrow pointing toward the right and within the light grey zone indicates a good match. An arrow pointing toward the left and within the dark grey zone indicates a poor match. The size of these colored zones is customizable, however, and the modeler can indicate the acceptance threshold for each zone. Each arrow represents a task of the process. The angle of the arrow represents the degree of knowledge matching for the given task. For example, the *Test Plan Writing* task (a horizontal arrow pointing to the right) is completely mapped, which means that all the concepts required for this task are provided, while the *UI Specification* task (a horizontal arrow pointing to the left) presents a complete mismatch between the prescribed concepts required to perform this task and the concepts available within the team resources. There can be as many zones as needed by the project.

4.1. Acceptable Match

The *Behavioral Model Creation* task, shown in Figure 3, is in the acceptable zone. This task is concerned with the creation of use cases, sequence diagrams, state diagrams, etc. To perform this task, a *modeling language* concept (C1) is required for both declarative and procedural knowledge. Declarative knowledge is needed to understand the theory behind this modeling language and procedural knowledge will enable the role to perform the task. Similarly, an *analysis technique* concept (C2) is required. A *guideline* concept (C3) is also required, since the diagrams produced must follow the templates of the organization. Basic declarative knowledge on the *requirement elicitation technique* (C4) is also required since the input of this task is the requirements document. The t'_T vector of this task is therefore:

$$\vec{t'_T}=\vec{C}_1\ (p,d)_{t'}+\vec{C}_2\ (p,d)_{t'}+\vec{C}_3\ (p,d)_{t'}+\vec{C}_4\ (d)_{t'} \quad (5)$$

It is found that this is not fully consistent with the concepts provided to the task. The roles are proficient in terms of the *modeling language* (C1), as they both have declarative knowledge and procedural experience. This is, however, a transformation task, requiring the transformation from requirements into diagrams. The roles are not familiar with this challenge, but hands-on training with a professional is planned. Declarative and procedural support for the *analysis technique* (C2) is therefore ensured. The requirements document presents declarative facts on the *guideline* to be used (C3). Is is also well enough detailed as to ensure that the *requirements elicitation technique* (C4) is well defined. The p'_E vector of this task is therefore:

$$\vec{p'_E}=\vec{C}_1\ (p,d)_{p'}+\vec{C}_2\ (p,d)_{p'}+\vec{C}_3\ (d)_{p'}+\vec{C}_4\ (d)_{p'} \quad (6)$$

The angle is then measured between the two vectors. Formula 7 shows the vectors in coordinate form, grouped by concepts, as well as the cosine and angle calculations.

$$\begin{aligned}
\overrightarrow{t'_T} &= \{(1,1)_{t'};(1,1)_{t'};(1,1)_{t'};(0,1)_{t'}\} \\
\overrightarrow{p'_E} &= \{(1,1)_{p'};(1,1)_{p'};(0,1)_{p'};(0,1)_{p'}\} \\
\cos\theta &= \frac{\parallel p'_E \parallel^2 + \parallel t'_T \parallel^2 - D^2}{2 \parallel p'_E \parallel \parallel t'_T \parallel} \\
&= \frac{6+6-1}{2*\sqrt{6}*\sqrt{6}} \simeq .92 \\
\theta &\simeq 22^\circ
\end{aligned} \tag{7}$$

This angle has been multiplied by two in Figure 3 to facilitate the visualization on a half-plane instead of a quadrant.

The measure shows that the concept matching is acceptable, albeit not perfect. The lack of procedural knowledge for the *guideline* concept implies that the diagrams might not follow the prescribed format appropriately. The main concepts are, however, correctly covered, which confirms that this task should not present a knowledge challenge.

4.2. Unacceptable Match

The task *Test Cases Redaction* shown in Figure 3 is in the inacceptable zone. This task builds test cases from the test plan and the architectural models for the application. Formula 8 presents the mismatch measure, where C1 is *testing technique*, C2 is *programming language*, C3 is *programming technique*, C4 is *modeling language* and C5 is *guideline*. Programming is a critical concept here because the test cases are written in the programming language of the application. Good tests must also be tailored to the application itself, requiring the primary role to read and understand the code.

The resource assigned to the role for the *Test Cases Redaction* task was not a competent programmer. This creates a serious issue for this task, which will probably create poor quality test cases, if anything at all. This problem is not obvious to the process designer as processes focus on the flow of data, and not on the knowledge required to handle this data.

4.3. Discussion on the Examples

These two examples show the risks faced by the project manager. An example presented by Jensen and Scacchi [18] shows that processes designed with only workflows in mind can overlook critical details. In their case, a process defined in a rich hypermedia presentation overlooks the workload of the Release Manager, resulting in a process bottleneck. The authors manage to find the problem through process simulation, but it could also be found through knowledge analysis. The Release Manager is required to produce many different documents but is not supported in this work. This will results in long delays between releases, as the Release Manager must acquire all required knowledge prior to producing each document.

5. Concluding Remarks and Future Work

This paper proposes a SPEM extension describing a new approach to concept mismatch identification in software process development. This approach enables the visualization of knowledge discrepancies in software process modeling. Such visualization provides new insights into key practices from the knowledge management viewpoint. An ontological approach coordinates knowledge representation in each SPEM element by linking the process model to one or more ontologies, according to the level of abstraction needed.

The concept matching indicator presents the state of the process model in terms of knowledge and the degree of deviation of each task within the process. This indicator could lead to more informed decision making; for example, in the recruitment of new competencies or the addition of more roles to support the primary role performing the task at hand. As a result, the knowledge-oriented view complements the

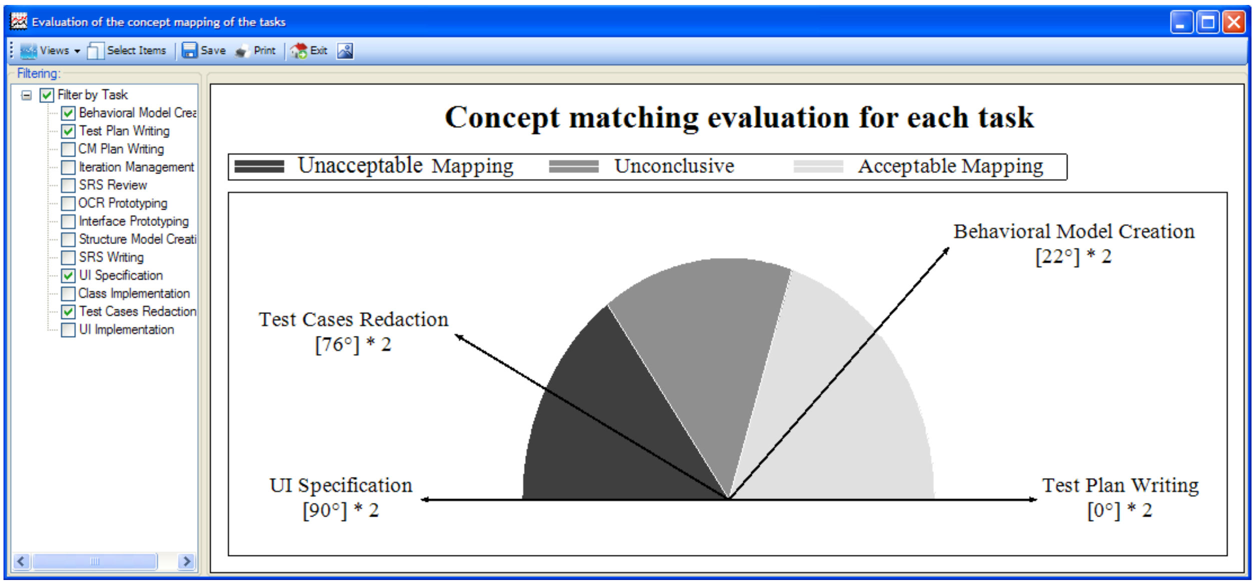

Figure 3. An example of the concept matching indicator

activity-oriented view, thereby fostering a better understanding of complex processes.

Future work will focus on organizational methods that support knowledge creation and propagation related to knowledge flows among software process models. The goal will be to study the propagation of knowledge throughout all the phases of the software process.

It will also seek ways to better integrate the expressivity of common ontological languages like RDF and OWL into our mismatch identification mechanism. This would expand our current tree-structured approach to resemble more flexible structures like the OWL-based SKOS structures.

Acknowledgment

We thank Olivier Gendreau for providing the project data used for our post-analysis. This work was partly supported by the "Fonds de Recherche sur la Nature et les Technologies" council of Québec (FQRNT) under Grant 127037 and NSERC grants A0141 and 361163.

References

[1] P.-N. Robillard, "The role of knowledge in software development," *Communications of the ACM*, Vol. 42, No. 1, January 1999, pp. 87–93.

[2] OMG, "Software & systems process engineering meta-model specification version 2.0," http://www.omg.org/spec/SPEM/2.0/, 2008, accessed: 10/Aug/2012.

[3] N. Kerzazi and P.-N. Robillard, "Multi-perspective software process modeling," in *8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2010)*, 2010.

[4] R. Martinho, D. Domingos, and J. Varajao, "Concept maps for the modelling of controlled flexibility in software processes," *IEICE Transactions on Information and Systems*, Vol. E93-D, No. 8, August 2010, pp. 2190–2197.

[5] T. H. Davenport and L. L. Prusak, *Working Knowledge: How Organizations Manage What They Know.* Harvard Business School Press, 1998.

[6] T. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation.* New York: Oxford University Press, 1995.

[7] J. D. Novak and A. J. Cańas, "The theory underlying concept maps and how to construct them," http://cmap.ihmc.us/publications/researchpapers/theorycmaps/theoryunderlyingconceptmaps.htm, 2000, accessed: 10/Aug/2012.

[8] D. P. Asubel, *The psychology of meaningful verbal learning.* New York: Grune & Stratton, 1963.

[9] Y. Wang, "On concept algebra and knowl-

edge representation," in *5th IEEE International Conference on Cognitive Informatics*, 2006, pp. 320–231.

[10] N. F. Noy and D. L. McGuiness, "A guide to creating your first ontology," Stanford University, Technical Report SMI-2001-0880, 2001.

[11] J. R. Anderson, M. Matessa, and C. Lebiere, "ACT-R: a theory of higher level cognition and its relation to visual attention," *Human Computer Interaction*, Vol. 12, No. 4, 1997, pp. 439–462.

[12] F. Bjørnson and T. Dingsøyr, "Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used," *Inf. Softw. Technol.*, Vol. 50, No. 11, October 2008, pp. 1055–1068.

[13] J. R. Anderson, *The Architecture of Cognition.* Harvard University Press, 1983.

[14] N. Anquetil, "Software maintenance seen as a knowledge management issue," *Information and Software Technology*, Vol. 49, No. 5, May 2007, pp. 515–529.

[15] N. Ferreira and R. J. Machado, "An ontology-based approach to model-driven software product lines," in *4th International Conference on Software Engineering Advances (ICSEA 2009)*, 2009, pp. 559–564.

[16] J. T. Nosek and I. Roth, "A comparison of formal knowledge representation schemes as communication tools; predicate logic vs semantic network," *International Journal of Man-Machine Studies*, Vol. 33, No. 2, August 1990, pp. 227–239.

[17] G. Salton and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, Vol. 18, No. 11, November 1975, pp. 613–620.

[18] C. Jensen and W. Scacchi, "Experience in discovering, modeling, and reenacting open source software development processes," in *International Software Process Workshop*, 2005.