How the Model Driven-Development May Improve Organizational Learning

André Menolli*
Department of Computer Science, Universidade Estadual do Norte do Parana, Brazil

Abstract
Organizational learning assists the companies to significantly improve their processes by means of experiences reuse, making knowledge accessible to the whole organization. In the software engineering area it is important that the knowledge is stored and systematically reused. Over the years, many researches have focused on improving organizational learning in software engineering area, addressing different techniques and topics. However, in software engineering always are being proposed new methods and techniques that can be applied to improve organizational learning. An instance is the Model-Driven Development, which allows the development of codes from high-level models. We think that Model-Driven Development may help improve some aspects of organizational learning, so we present a brief reflection about this issue.

ABBREVIATIONS
MDD: Model-Driven Development; OL: Organizational Learning

INTRODUCTION
The capacity to create, to acquire, to integrate, to implant and to distribute knowledge has appeared as a basic organizational capacity [1], and currently with the growing amount of information that companies deal, it is continually increasing importance of knowledge management. Hence, competitive companies should manage and explore the whole knowledge as strategic options for future decisions and competitive advantages [2].

Furthermore, the knowledge should be used not just to give competitive advantages, but also to improve performance based on the experience, refining the organizational processes and avoiding past errors, this set of procedures are defined by the concept of organizational learning [3]. Moreover, for some kinds of companies the organizational learning is essential, mainly for knowledge-intensive companies such as those involved with software-development. However, for this kind of companies, organizational learning is more difficult to apply, since is hard to acquire and organize past experiences, because most of the time the knowledge is internalized in the minds of employees, making difficult the knowledge sharing.

In software engineering area, many works proposes approaches, methods and tools to minimize the problems caused by the misapplication of organizational learning, as shown in the work [4], which exposed that in all software engineering knowledge areas there are initiatives to improve organizational learning. Nevertheless, despite the large numbers of works that focus in this direction, in software engineering always are being proposed new methods and techniques, and they can be applied to improve organizational learning.

A topic that is not very explored, and has great potential to assist organizational learning is the Model-Driven Development (MDD). The MDD allows the development of codes from high-level models and their function is essentially to turn a software model into executable code, be fully or partially varying according to the needs. Thus, we present a reflection about how MDD can support improve organizational learning.

BACKGROUND
In this section, we present a brief review of the main topics involved in this work: organizational learning, which specifically we focus on this topic applied to software engineering; and MDD, which is presented a brief introduction.

Organizational Learning and Software Engineering
Several companies around the world attempt to improve their processes based on previous experiences. These companies are looking for achieve some level of OL, even though they often are not aware of it. To reach OL, companies can use different knowledge management techniques, depending on the type of knowledge created and what company intends to do with it.

OL can be defined as the continuous testing of experience and
its transformation into knowledge that is accessible to the whole organization and relevant to its basic purposes [5]. Thus, OL is an important factor in software development, especially because as mentioned previously, software projects are by nature knowledge-intensive [6].

Consequently, due to such particularity, there are some initiatives to improve organizational learning that are inherent in software engineering area. So, when the learning and management approach is used, the knowledge created during software processes can be captured, stored, spread and reused [7]. Thus, better quality and productivity can be achieved.

Among the main initiatives in software engineering area, we can highlight the study of Basili et al. [8], which focuses in to maintain an appropriate level of knowledge in the organization. The study of Basili et al helped to promote research into knowledge management in software engineering and to create and popularize new terms and concepts such as 'experience base' and 'experience management'. Over the years, new terms were defined and consolidated for managing experiences, such as 'experience factory' and 'lessons learned' (LL), which were subsequently extensively adopted in software-engineering knowledge management studies.

An area of software engineering that has been explored in recent years is agile development. As example, we may cite the work of Livari et al. [9] that analyzed the relationship between organizational culture and the post-adoption deployment of agile methods, and [10], in which the authors try to understand the challenges of shared decision-making in agile software-development teams. Agile software development requires alignment of decisions at the strategic, tactical and operational levels to overcome these challenges as well as a transition from specialized skills to redundancy of functions and from rational to naturalistic decision-making.

Furthermore, a broad systematic review was undertaken to identify which software-engineering areas OL studies are concentrated in. Moreover, this study considered how and what OL concepts of the main theories, as Kolb’s model of experiential learning [11], the double-loop learning theory proposed by Argyris and Schön [12], Wenger’s communities of practice theory [13] and Nonaka and Takeuchi’s theory of knowledge creation [14], were applied in software engineering in recent years [4].

Summarizing, companies have ays used technologies to improve OL, nonetheless new technologies and methods always are being proposed in software engineering area, hence new possibilities to improve OL emerge with them.

**Model-Driven Development**

Since the last decade, Model Driven Development has evolved significantly due to its flexibility and applicability [15]. Model-driven development, also known as MDD, is a software-engineering approach aiming to raise the level of abstraction of an application and, hence, simplify and formalize software life cycle stages and tasks by using models and model technologies [16]. MDD is primarily concerned with reducing the gap between problem and software implementation domains through the use of technologies that support systematic transformation of problem-level abstractions to software implementations [17].

MDD regards the use of models in the software development life cycle and states that MDD automates software development via model processing, model transformation, and code generation techniques. Conventionally, developers employ two different approaches for software development [15]:

- Design solutions visually; and
- Code solutions right straight based on functional requirements.

Despite both approaches may present advantages and disadvantages, MDD tends to bring simultaneously the better of each one of them. By the way, through MDD, developers are able to not only design their solutions, but also and build partial artifacts which will be part of the final product. Furthermore, the term “model driven” may be considered by some to be redundant in Model-Driven Engineering (MDE) given that engineering of software invariably involves modeling. While this may be true, it is currently the case that software developers seldom create and effectively utilize models other than code. The term “model driven” in MDE is used to emphasize a shift away from code level abstractions [17].

Likewise, Selic [18] states that the main focus of MDD regards models rather than computer programs. Furthermore, Selic [18] points out the expression of models using concepts that are not strictly associated with low level languages, although close to the real domain of the problem, as the main advantage of adoption MDD.

**Using MDD to improve Organizational Learning**

The MDD may be a good strategy to improve organizational learning, since it uses high-level models, called Meta Model, focuses on creating and exploiting domain models. These conceptual models describe all topics related to a specific problem, and generally is defined by an expert. All implementations of the problem are performed based on these conceptual models.

Currently, different model transformation tools for a variety of purposes have been explored in the literature, for instance, XSLT (Extensible Stylesheet Language Transformation), QVT (Query-View Transformation language), ATL (ATLAS transformation language) and Acceleo [19]. The Figure 1 presents a generic model, showing how a MDD implementation using Acceleo works. Acceleo is an implementation of the Object Management Group (OMG) Model to Text Language (MTL) standard. The OMG, in its role as an industry-driven organization that develops and maintains standards for developing complex distributed software systems.

The main idea of MDD is that models are (semi)automatically transformed into executable code by model transformations. In a MDD solution, generally there are two roles:

- An Expert: responsible for creating the MetaModel and Model Transformations;
- A Developer: responsible for creating an application model based on the MetaModel.

Analyzing the (Figure 1) it is noticed that developers create specific models for their context based on MetaModels. Moreover, the expert is responsible for defining, besides the MetaModel,
the transformation model, which describes how source code is generated based on parameters set on MetaModel and the Application Model. Therefore, the code generator requires an Application Model and a Transformation as input for generating low-level source code as output.

Summarizing, the MDD approach is intended to increase productivity and may help to codify tacit knowledge in high-level models; further assist developers use these models. The MDD assists the code reuse (via reuse of standardized models) and simplifying the process of design. Furthermore, MDD may promote a standardization of the terminology and the best practices used in the application domain. Hence, MDD is considered effective if its models make sense from the point of view of a user that is familiar with the domain, and if they can serve as a basis for implementing systems.

DISCUSSION

Many previous studies have focused on improving organizational learning through enhancing the documentation process or the software development process. The studies that focus on improve the development process, normally are concerned about the definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle processes themselves, and it is common that they propose tools to help these tasks.

Furthermore, concepts as Postmortem Analysis and Experience Factor are frequently used in software engineering studies applied to organizational learning. These concepts are very important; however, they provide little benefit in order to codify tacit knowledge efficiently. On the other hand, MDD is able to assist codify tacit knowledge in high-level models, besides helping in the software development standardization. Therefore, MDD is a new option to improve organizational learning, and should be used together with other consolidated approaches to capture the knowledge.

Additional analysis that can be done, it is understand how the MDD may be related with organizational learning theories. For this purpose, we use one of the most accept organizational learning theory, which is known as SECI [14], that describes a spiral model in which learning generates new knowledge within a company by means of the interaction between tacit and explicit knowledge. Nonaka and Konno [14] identified four forms of knowledge conversion: socialization, combination, internalization and externalization. Among the four forms of knowledge conversion presented by Nonaka and Konno, the use of MDD clearly may assist in three of them, but more strongly in two: Externalization is the process of converting tacit knowledge into explicit knowledge by means of metaphors, analogies, concepts, hypotheses or models. When MDD is used, externalization is exactly one of the results produced, since tacit knowledge is codified into models.

Combination is the conversion of explicit knowledge generated by an individual to add it to the explicit knowledge of the organization, thereby generating a type of knowledge called “systemic knowledge”. When an expert creates a metamodel, that represents a standard of some domain, this knowledge become part of organization. This new explicit knowledge is then disseminated among the members of the organization through the reuse model.

Although, MDD acts more strongly on externalization and combination, it may acts in internalization. Internalization is the conversion of explicit knowledge into tacit knowledge. The process is closely related to learning by practice and generates a type of knowledge called “operational knowledge”. When a metamodel is created, this knowledge that is often inaccessible to some people, become accessible. Using a metamodel to develop an application, the developer increase their skill on the
topic, because he needs understand the metamodel, this helps to improve his knowledge about the issue.

FINAL CONSIDERATIONS

Organizational Learning is an interest topic for software engineering, mainly considering the nature of knowledge generated during the software development process. Due to this, OL is a topic explored a long time ago in this area, however new possibilities to improve it are always emerging. One of these possibilities is the MDD; we consider it a promisor topic to help in the OL process. Although this work presents a preliminary reflection, it has shown that MDD may support concepts described in an important OL theory. To gain a better understanding of how the MDD is related to other theories, it is necessary to carry out a more in-depth study.

REFERENCES