Knowledge Transformation in Software Development Processes

Ilia Bider
Department of Computer and Systems Sciences (DSV), Stockholm University (SU)
IbisSoft AB
Stockholm, Sweden
ilia@{dsv.su.se | ibissoft.se}

This position paper discusses the theoretical foundation of Software Engineering (SE) and argues that it should consist of two branches, the first branch deals with the generic properties of software products, the second one deals with the generic properties of SE processes. The paper argues that one of the important properties of the software development process is its model of knowledge transformation, and it suggests to adapt Nonaka’s SECI model for investigating and comparing various software development methodologies.

Software Engineering; knowledge transformation; SECI model; software development; agile

I. INTRODUCTION

Software Engineering (SE) is an engineering discipline that deals with manufacturing of certain kind of products – software. Therefore, the SE theoretical foundation should concern both:

• the properties of the product (software), and
• the properties of the processes of product development, maintenance and retiring/disposing

This means that the theoretical foundation of SE is naturally split into two main areas, which, while being interconnected, have different underlying concepts.

As we deal with the theoretical foundation, the properties of both products and processes should be considered on the high abstract level, so that the theory can be applied to any kind of software, and to any kind of methods of software development and maintenance.

A typical example of product properties on the high abstract level is Software Quality, the subject addressed in numerous research works. Less popular subject that, in our view, is of great theoretical interest is multilayered architecture of software products. We believe that the theoretical foundation for creating a product-independent platform to address this subject can be found in the works of Michael Polanyi [1]. In it, he introduces the principle of boundary control according to which each layer has two sets of laws to obey. One set is the laws of this layer, the other set is the laws of boundary control from the layer above which uses this layer for attaining its own goals/properties. In addition, he states that the laws of the upper layer are not reducible to the laws of the lower one.

In this paper, we, however, leave the properties of the software products outside the scope of our consideration and concentrate on theoretical foundation for the software development process. As this is an intellectual process which practically does not deal with the physical world, the transformation of knowledge in this process is of major importance. There are numerous methods of software development; therefore creating a theoretical foundation for knowledge transformation that is applicable to any of them is a challenging task. We propose to use ideas from Nonaka’s SECI model [2] as a foundation for the theory of knowledge transformation in the software development process. In this short paper, we will give an outline of how ideas from SECI could be applied to software development.

II. SECI MODEL

SECI, which stays for Socialization – Externalization – Combination – Internalization, see Fig. 1, was developed in [2] to explain the ways how knowledge is created in an organization while being transformed from the tacit form to explicit and back.

In the sections that follow, we demonstrate how the ideas from SECI can be applied to software development process.

III. KNOWLEDGE TRANSFORMATION IN THE TRADITIONAL SOFTWARE DEVELOPMENT

Knowledge transformation during the traditional cycle of software development is represented in Fig 2. It starts with tacit knowledge on the needs that exists in the heads of the stakeholders. Then, it is transformed into explicit knowledge of requirements specifications which correspond to the...
**Externalization** in Fig. 1. After that, this explicit knowledge is converted into another explicit form of design specifications. This transformation corresponds to **Combination** in Fig. 1, as the transformation is done with the help of software design principles of the appropriate SE domain. The next step, **Coding**, consists of transforming explicit knowledge of design specifications into the knowledge embedded into a software system. Though parallel to **Internalization** from Fig. 1, this step does not correspond to the latter exactly. We call this step **Embedment**. The next step, **Learning to use**, consists of transforming the knowledge embedded in the software system into the tacit knowledge of its users that use the system in their practice. Though parallel to **Socialization** from Fig. 1, this step does not correspond to the latter exactly. We call this step **Adoption**.

The risks above can be minimized by employing qualified requirements engineers, developers, programmers, and trainers. However, they can never be totally eliminated. The biggest risk of all, however, in today's highly dynamic environment is that:

1. Requirements do not catch the needs properly
2. Requirements are not converted into a proper design
3. Coding does not follow the design exactly
4. The new software is not properly understood by its users, and it is rejected or used in the wrong fashion
5. While a new system is under development, the problems/needs are continuing to evolve. As the result, a wrong/outdated system is delivered to the stakeholders.

**IV. KNOWLEDGE TRANSFORMATION IN THE AGILE SOFTWARE DEVELOPMENT**

One way to minimize the risks of the traditional software development is to use the agile principles [3]. Knowledge transformation in the agile development in the idealized form is represented in Fig. 3. In it, the **Design** phase is removed, and one big cycle is substituted by many small ones. This corresponds to the main idea of agile development to avoid, as much as possible, transforming knowledge into explicit form. Minimum requirements and design documents, e.g., notes, emails, or black-whiteboard diagrams, however, still exist, but they do not represent legally binding documents of the traditional approach.

As we see from Fig. 3, the nature of the requirements engineering phase is also changed. It consists in transferring tacit knowledge of problems/needs from the stakeholders to the design team, and thus corresponds to **Socialization** in Fig. 1. As the result we get the repeating cycle of **Socialization**-**Embedment**-**Adoption** – **SEA model**.

**V. CONCLUSION**

As we discussed in Section I, the theoretical foundation of SE should have two branches, one concerns the generic properties of software, the other concerns the properties of SE processes. As there are multiple different methods of developing software, the theory of the second branch should be able to model different sides of these methodologies, so that they could be compared, and the one that suits best a given context of development could be chosen.

We believe that SE theory should include an approach to describing and modeling knowledge transformation in software development processes. As we have shown in Section II-IV, SECI model from [2] could be adapted for this kind of modeling.

The ideas presented in this position paper are based on the analysis of the authors own practice of developing software systems (agile as well as not agile) and introducing them into organizational practice, see for example [4].

**REFERENCES**