10-2008

THE KNOWLEDGE-GAP REDUCTION IN SOFTWARE ENGINEERING

Salem Ben Dhaou Dakhli
Paris-DauphineUniversity, France, sdakhli@computer.org

Ben Chouikha Mouna
Paris-DauphineUniversity, France, mouna.benchouikha@dauphine.fr

THE KNOWLEDGE-GAP REDUCTION IN SOFTWARE ENGINEERING

Dakhli, Salem Ben Dhaou, Paris-DauphineUniversity, Place du Maréchal de Lattre de Tassigny, 75016 Paris, France, sdakhli@computer.org

Ben Chouikha, Mouna, Paris-DauphineUniversity, Place du Maréchal de Lattre de Tassigny, 75016 Paris, France, mouna.benchouikha@dauphine.fr

Abstract

Many papers proposed in the software engineering and information systems literature are dedicated to analysis of software projects missing their schedules, exceeding their budgets, delivering software products with poor quality and in some cases even wrong functionality. The expression "software crisis" has been coined since the late 60's to illustrate this phenomenon. Various solutions has been proposed by academics and practitioners in order to deal with the software crisis, counter these trends and improve productivity and software quality. Such solutions recommend software process improvement as the best way to build software products needed by modern organizations. Among the well-known solutions, many are based either on software development tools or on software development approaches, methods, processes, and notations. Nevertheless, the scope of these solutions seems to be limited and the improvements they provide are often not significant. We think that since software artifacts are accumulation of knowledge owned by organizational stakeholders, the software crisis is due to a knowledge gap between resulting from the discrepancy between the knowledge integrated in software systems and the knowledge owned by organizational actors. In particular, integrating knowledge management in software development process permits reducing the knowledge gap through building software products which reflect at least partly the organization's know-how. In this paper, we propose a framework which provides a definition of knowledge based on information systems architecture and describes how to deal with the knowledge gap of a knowledge oriented software development process which may help organizations in reducing the software crisis impacts.

Keywords: knowledge, software process, organizational actor, business function, business entity, business process, information.

1 INTRODUCTION

Many papers proposed in the software engineering and information systems literature are dedicated to analysis of software projects missing their schedules, exceeding their budgets, delivering software products with poor quality and in some cases even wrong functionality. The expression “software crisis” has been coined since the late 60’s to illustrate this phenomenon. Various solutions has been proposed by academics and practitioners in order to deal with the software crisis, counter these trends and improve productivity and software quality. Such solutions recommend software process improvement as the best way to build software products needed by modern organizations. Among the well-known solutions, many are based either on software development tools or on software development approaches, methods, processes and notations. The Unified Process, the UML notation, the Object-Oriented paradigm are examples of solutions proposed since the end of the 80’s. Nevertheless, the scope of these solutions seems to be limited and the improvements they provide are often not significant. Indeed, the well-established software development approaches, methods and processes are based on a rational mechanistic view of organizations. According to this view, to be efficient software development must be compared to traditional manufacturing processes which are routinized using the Taylorian scientific management principles. Such a view is criticized by many authors for many reasons. Firstly, it is technology oriented, reductionist and does not reflect the complexity of the modern organizations reality. Secondly, it is based on the procedural rationality concept which assumes that to solve a problem, organizations look for the best solution called silver bullet by (Brooks 1987). On the basis of the Bounded Rationality theory proposed by (Simon 1983), Brooks demonstrates that there is no miraculous silver bullet and that organizations solve the software engineering problems they encounter by building satisfycing (i.e. good enough) solutions (Brooks 1987) (Brooks 1995). Moreover the systematic, disciplined and quantitative view of software engineering induced by the well-known and approaches methods does not take into account all the dimensions of software in particular organizational, economic, and human dimensions (Toffolon 1999) (Ilavarsan et al. 2003). (Boehm 2006) stresses that the software engineering discipline is multidisciplinary and oriented toward people. Therefore, software engineering relies not only on computer science but also on economics, behavioral sciences and management sciences. (Turner et al. 2003) suggest the weaknesses of the methods and approaches - proposed during the last four decades to deal with the software crisis – are related to the bureaucratic project management process they induce. These authors use the economic agency theory (Alchian et al. 1972) (Jensen et al. 1976) to analyze software projects as a temporary organization similar in complexity, uncertainty and risks to modern organizations. According to this point of view, the management of software projects is characterized by conflicting relationships between organizational actors linked by principal/agent contracts. Therefore, the use of project rational and systematic management methods induced by the well-established software engineering approaches and methods is not appropriate and results in software projects failures associated with the software chronic crisis. This analysis is not efficient for addressing the complexity of modern organizations since the economic agency theory is rooted in the procedural rationality paradigm. We think that since software artifacts are accumulation of knowledge owned by organizational stakeholders (Baetjer 1998), the software development process must be knowledge-oriented. In particular, integration of knowledge management in software development process permits building software products which reflect at least partly the organization’s know-how. In this paper, we propose a framework of a knowledge oriented software development process which may help organizations in reducing the software crisis impacts. Our framework is based on a joint meta-model common to business processes architecture, information systems (functional) architecture and applicative architecture. In this framework, we define knowledge owned by organizational actors using the main concepts identified by this meta-model. This paper is organized as follows. In section 2, we present the meta-model on which relies the knowledge-oriented development process proposed in this paper. Section 3 describes a meta-lifecycle of this process and emphasizes its knowledge-oriented nature. In section 4, we conclude this paper by listing the problems encountered and the future research directions.

2 THE PROCESS-APPLICATION-FUNCTION (PAF) META-MODEL

(Dakhli 2008) proposes a software solution architecture multi-layered model which relies on five interacting layers: the strategy layer, the functional architecture (information system architecture) layer, the applicative architecture layer, and the software architecture layer (Figure 1). The strategic layer defines the organizational problems to be solved and their organizational solutions. Such problems result from the organization's external and internal constraints. External constraints may be economic, political, social, legal or related to the evolution of the technology. Internal constraints reflect the impacts of external constraints on the organization's components: structure, people, production technology, tasks and information technology (Toffolon 1996) (Leavitt 1963).

The business process layer describes the business processes architecture at the conceptual and the organizational levels. The business processes conceptual architecture models business processes as a nexus of activities exchanging and processing information. The organizational business processes architecture is the projection of the conceptual business processes architecture on the organization. Therefore, it models business processes as nexus of operational activities and tasks carried out by organizational actors in order to create value. The business processes architecture is updated according to the organizational solutions defined by the strategic layer.

The functional architecture layer describes the information system architecture as a nexus of business entities and functions. A business entity is a set of information chunks which define a concept used by the organizational actors while carrying out a business process. A business function is an action which uses and transforms at least one business entity. A business process manipulates business entities through the use of business functions. Business entities are described in a business repository. A business function may be considered as an aggregation of many business sub-functions. Business functions may be used by many business processes. Such business functions are called reusable business functions. Business entities manipulated by many business processes are called shared information. Because of the invariant and stable nature of business entities and business functions, they are independent of the organizational structure and the roles played by actors within an organization. Architecture of an organization's information system is defined as a model describing the organization's business functions and business entities as well as the relationships between these concepts. The business processes architecture is updated by integrating the impacts of the organizational solutions defined by the strategic layer on the business entities and functions.

The applicative layer provides a map which describes the organization's applications as well as the information flows they exchange. An application is a set of software systems which computerizes at least partly a business process. So, an application provides a software support to the value creation behavior of organizational actors. This behavior consists in carrying out business processes activities which manipulate business information by using business functions. An application provides two categories of services: service-to-user and service-to-application. A service-to-user results from an interaction between and end-user and an application and helps an organizational actor who carries out a set of operational activities. A service-to-application is an intermediate service provided by an application to another application while processing information. An application may be considered as a dynamic conjunction of a set of business process activities with business entities and business functions in order to contribute to goods and services production. The applicative layer results from the interaction between the functional layer and the business process layer which supports the problem and operation spaces. The applicative layer delivers a first level description of a software solution as a new or enhanced application which interacts with existing and future applications.

The software layer describes each software solution as a set of software components and connectors distributed according to a software architecture model (e.g. MVC,…). A software solution is either the architecture of a new application which supports at least partly a new business process or the architecture of an existing application which is enhanced in order to take into account the modifications of an existing business process. Despite the richness of the existing definitions of the software component concept, we think that these definitions are note appropriate to take into account all the perspectives of information system architecture. So, we propose in this paper a definition of this concept which refers to business functions. Our definition states that a software component is an autonomous and homogeneous logical unit which implements a business function in order to provide a

service either to end users or to other logical units. A software connector is an autonomous and homogeneous logical unit which facilitates interactions between two software components. A software solution is composed of reusable and specific software components and connectors. A reusable software component implements a business function used by many business processes.

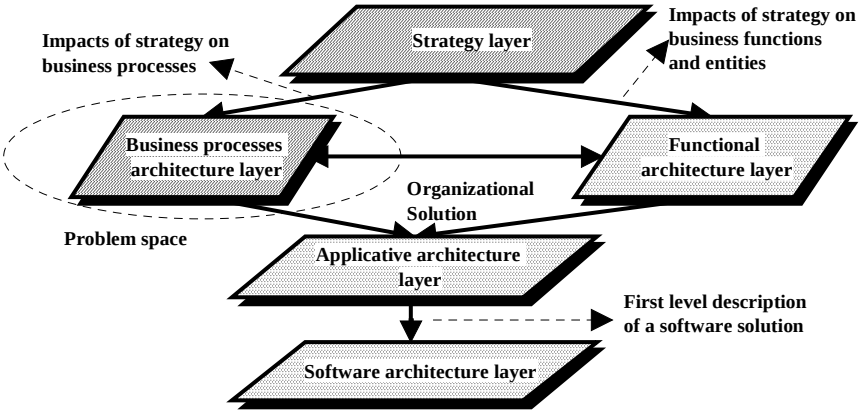

Figure 1: The software solution architecture multi-layered model [Adapted from (Dakhli 2008)]

Consequently, the software solution architecture has many facets associated to the four layers presented above. Each facet corresponds to an architecture meta-model which describes the basic concepts characterizing this facet and their relationships. The main concepts used by the four meta-models are interrelated as illustrated by the **PAF** global meta-model of software solution's architecture (Figure 2).

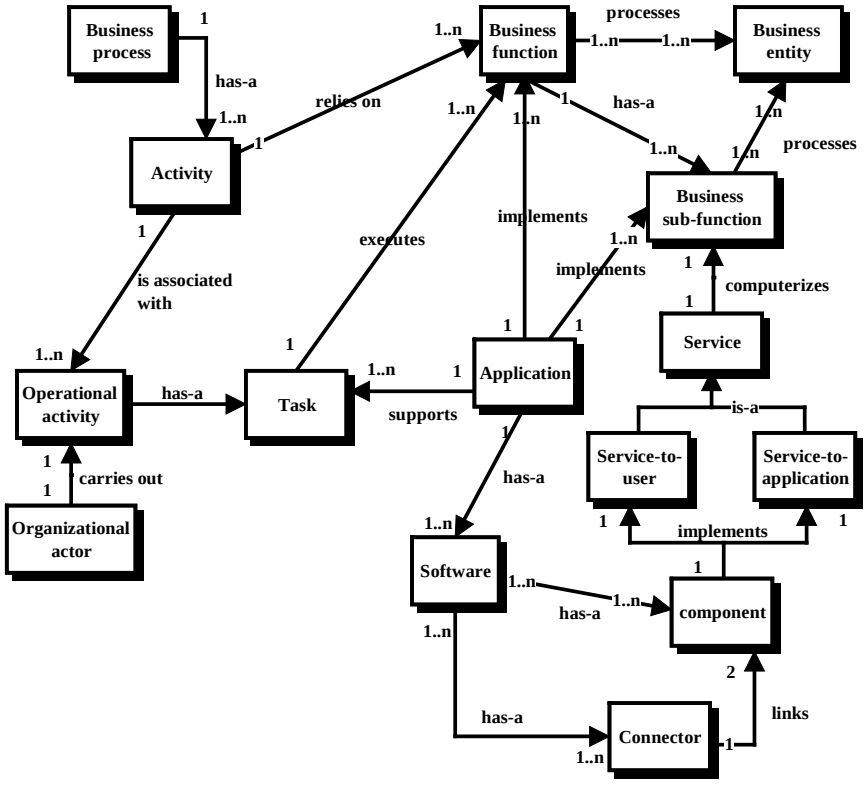

Figure 2: The **PAF** meta-model of software solution's architecture

3 THE KNOWLEDGE-ORIENTED SOFTWARE DEVELOMENT PROCESS META-LIFECYCLE

Prior to the presentation of our knowledge-oriented software development process meta-lifecycle, we define the concept of knowledge used in this work and analyze how software systems integrates knowledge.

3.1 The concept of knowledge

Knowledge is defined as information that is relevant for executing certain business actions. According to (Nonaka et al. 1995) and (Newell et al. 2001), knowledge may be understood as a justified true belief. This definition emphasizes the temporary nature of knowledge. The knowledge and information concepts are mutually dependent. On the one hand, information is external to human beings and is stored in various supports like books or databases while knowledge in internal to the minds of knowledge workers. On the other hand, information is converted to knowledge through the internalization process and knowledge is transformed into information once it is externalized (Alavi et al. 2001). Internalization consists in transforming information which is external into knowledge which is internal to the minds of knowledge workers. Learning and information processing by knowledge workers facilitate internalization and result in creation of new knowledge, or alteration of existing knowledge. Externalization consists in articulating internal knowledge in order to making it external to the mind of a knowledge worker. Externalized knowledge is called explicit knowledge or information while tacit knowledge is knowledge which cannot be articulated (Zack 1999). Writing a book is an example of externalization of knowledge owned by an author. Knowledge is tacit if it is difficult to express using some understandable symbols like written notations or spoken language. The concept of tacit knowledge was introduced by (Nonaka 1994), drawing on the more philosophical work of (Polanyi 1967) which considers that tacit knowledge is the type of knowledge we use to carry out the actions that we perform routinely without thinking consciously about how to carry these actions.
Another important characteristic of knowledge is that it is related to action (Nonaka 1994) (Nonaka et al. 1995) (Buckley 2001). This means that the value of knowledge results from the ability of knowledge workers to impact the real environment in which they operate. Moreover, the action of knowledge workers within the environment where they operate generates feedback information which facilitates organizational learning. In particular, tacit knowledge can take the form of embodied knowledge which is materialized by action of knowledge workers (Blackler 1995). In this paper, we define knowledge as the interaction between organizational actors, business processes, business functions, and business entities. Therefore, knowledge determines how an organizational actor contributes to value creation when he carries out tasks associated with his role within an organization. This definition suggests three remarks. Firstly, it underlines the dependence of knowledge owned by an organizational actor on how he perceives tasks and business functions. Secondly, it recalls that organizational actors create knowledge by processing information. Finally, this definition takes into account the relationships between knowledge and action as well as the temporary nature of knowledge. Indeed, interaction between an organizational actor, a business process, business functions and business entities depends on existing knowledge owned by this actor. Such knowledge is continuously updated through processing of information provided by the organizational environment and the business entities manipulated during the organizational actor action.

3.2 The knowledge gap analysis

Since software systems are accumulation of knowledge (Baetjer 1998), they may disseminate knowledge not only contained in books and stories but also owned by knowledge workers who carry out business, support and decision-making processes within organizations. Moreover, knowledge integrated in software systems is made operational to solve problems encountered by organizational actors who operate within organizations. The effectiveness of knowledge integration in a software system determines the quality of this system i.e. how they support business, support and decision-

making processes within organizations. Moreover, the difference between the quality of two software systems results from the amount of knowledge they disseminate. This is because knowledge integrated in software system determines their ability to support effectively organizational processes. Consequently, a software system may be thought as knowledge container whatever the method, the language or the technique used to develop it. This point of view provides us with a different explanation of the roots of the software crisis. The knowledge gap is the main reason of the rejection of a software system by the organizational actors it is intended for. Such a gap consists in the difference between the knowledge integrated in a software system and the knowledge owned by the organizational actors who use this system while carrying out their activities. Furthermore, the knowledge gap associated with a software system is dependent on the organizational actors who use it. This means that the value provided by a software system to an organizational actor using it depends on the amount of knowledge proper to this actor and integrated in this software system. So, there are many knowledge gaps associated with a software system, each gap expresses the discrepancy between the knowledge owned by an organizational actor and the amount of knowledge integrated in the software system. Consequently, an organizational actor rejects a software system if this system doesn't reflect a sufficient part of the knowledge owned by this actor. Besides, the knowledge gap may be targeted as the root cause of the users resistance (Hirschheim et al. 1988) (Toffolon 1996) (Dakhli 1998). Following the terminology of (Brooks 1987), the knowledge gap seems to be an essential difficulty associated with software engineering. Nevertheless, even if there are many organizational actors who consider that a software system integrates a part of their knowledge, the acceptance of such a system is not easy, as it is not restrict itself to providing a set of services, but induces changes in the organization and balance of knowledge. Therefore, the knowledge gap has two facets: conservative and extensive. The conservative facet of the knowledge gap describes the impact of a software system on the organization and the balance of knowledge owned by an actor or exchanged by many actors. The extensive facet of the knowledge gap provides an answer to the following question: what is the part of the knowledge owned by an individual actor is not integrated by a software system used by this actor?

During the last two decades, many authors have stressed that requirements engineering provides the appropriate solutions to reduce the impacts of the software crisis (Wiegers 2003) (Neill et al. 2003) (Davis 2005) (Regnell et al. 2005) (Glinz et al. 2007) (Dieste et al. 2008). Nevertheless, the solutions proposed by the requirements engineering community do not provide instruments effective enough to deal with the software crisis in particular because these solutions take into account the knowledge gap. This means that solutions to the software crisis based on requirements engineering don't describe how to gather, combine, transform, and integrate knowledge into a software system through a sequence of specifications with an increasing degree of formalism. Therefore, to improve software systems quality through the knowledge gap reduction, the software development process core activities must be dedicated to cooperative and iterative knowledge engineering (Toffolon et al. 2007).

3.3 Integration of knowledge in software systems

In this work, the expression "knowledge engineering" refers to a set of activities related to knowledge gathering, storage, combination, transformation and transfer. According to (Toffolon et al. 2007), the cooperative nature of the knowledge engineering process is due to two types of asymmetries: know-how asymmetry and understanding asymmetry. Know-how asymmetry is related to tacit and articulated knowledge owned by human actors and originates from the dispersion of knowledge across stakeholders and existing software artifacts. For example, there is a know-how asymmetry between organizational actors belonging respectively to the problem side (end user, customer) and the solution side (architect, developer). The customer and the end user are domain experts who understand the practice and know implicitly what the system is supposed to do. They do not know the technological possibilities for supporting their work. The architect and the developer know how the technology can do it but they ignore whether the technology they create will be appropriate for the support of operational and decision processes. Understanding asymmetry results from the differences between stakeholders understanding of knowledge disseminated in existing software systems and their perspectives of what the future software system should be.

The reduction of the knowledge gap requires building a common vision of the future software system shared by all the organizational actors concerned with this system. The know-how and understanding asymmetries constitute obstacles to this challenge since they contribute to the knowledge gap aggravation. That is the reason why the software engineering activities and the knowledge engineering activities they embed must be cooperative i.e. stakeholders have to work together in order to reduce asymmetries and build a common view of the required future software system. Such a view is based on knowledge embedded in existing software artifacts or owned by stakeholders combined and transformed through the knowledge engineering process. Effective cooperation of stakeholders results in software prototypes which either permit uncertainty reduction or are pieces of the final software system. In the first case, informative prototypes are built to extract knowledge embedded in existing artifacts or owned by stakeholders and to illustrate a common understanding of requirements and needs. In the second case, final versions of software modules, called operational prototypes define and implement the stakeholder's common view of what the final software system should be. Operational prototypes are parts of the software system version delivered to end-users. Informative prototypes reduce uncertainty inherent in requirements and generated by know-how and understanding asymmetries. They may be considered as communication tools which facilitates knowledge transfer between all the stakeholders involved in software systems development, maintenance, and use. Operational prototypes are dependent on informative prototypes which provide them with knowledge shared by stakeholders and necessary to build a common vision of the future software system. Such a common vision may be considered as the smallest common denominator synthesizing knowledge shared by all the stakeholders. The iterative nature of the software engineering process and the knowledge engineering activities it embeds stem mainly from the volatility and the fuzziness of stakeholders' requirements. During each iteration, informative prototypes are built, discussed and assessed by stakeholders working together prior to developing a version of the final software system composed of operational prototypes. Therefore, each version of the final software system, issued from an iteration of the software development process, reflects the state of the vision of the software problem and solution shared by stakeholders. Besides, an evolution of the stakeholders shared vision of the problem and the solution often results in an evolution of a software system integrating this updated common vision. Consequently, building a shared vision of the software problem and the software solution - i.e. the future software system required by an organization – conditions the approval of the future software system by the concerned organizational actors. In the next subsection, we present a meta-lifecycle on which rests the process which permits building of the shared vision to be integrated in future version of software systems.

3.4 The meta-lifecycle of knowledge integration in software systems

The proposed meta-lifecycle of knowledge integration in software systems is based on the four-staged **DUCA** (**D**iscover-**U**nderstand-**C**onstruct-**A**ssess) lifecycle due to (Toffolon et al. 2007). In this paper, we contribute to improvement of this lifecycle by using the **PAF** (Process-Application-Function) meta-model presented in a previous section to describe how the **DUCA** four stages take place. According to (Toffolon et al. 2007), each iteration of the knowledge-oriented software development process rests on the **DUCA** lifecycle which is composed of four stages (Figure 2):

❶ Discover the problem knowledge i.e. knowledge embodied in existing software artifacts and other internal and external sources (books, guides, …) or owned by stakeholders

❷ Understand the body of knowledge issued from the Discovery stage and build a shared vision of the problem knowledge

❸ Construct a common vision of the software solution

❹ Assess the software solution

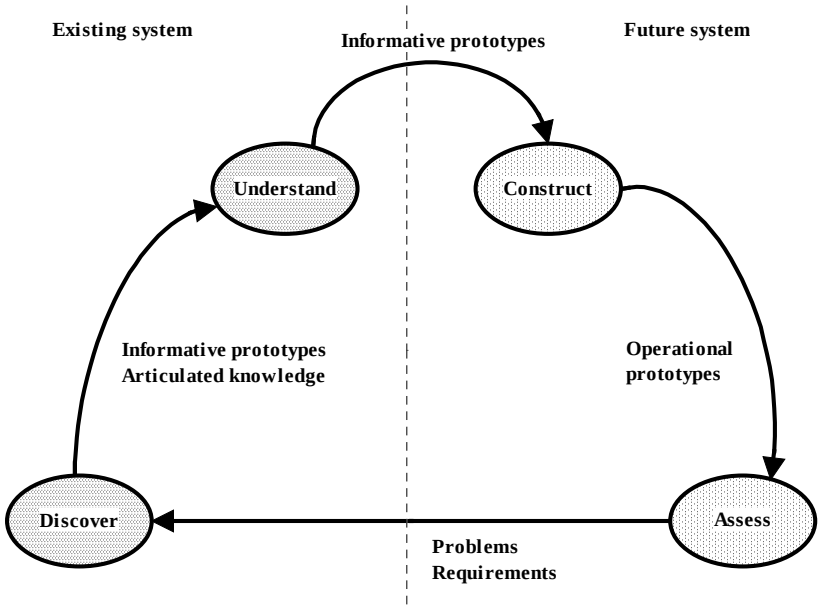

Figure 2: the **DUCA** lifecycle [Source (Toffolon et al. 2007)]

3.4.1 The Discovery stage

The Discovery stage consists in gathering knowledge related to problem and the software solution. Such a knowledge is owned by stakeholders or embodied in existing software artifacts and other external and internal sources. In addition to information stored in various supports like written guides and databases, the knowledge discovered at this stage is explicit knowledge externalized by the stakeholders during brainstorming sessions or interviews. The output of this stage splits into two categories: static and dynamic. Static knowledge refers to information chunks which describe the organizational solution and the software problem. Dynamic knowledge has three components. On the one hand, it includes a description of business processes, support processes and decision-making processes to be supported by the future software solution. On the other hand, it includes a description of organizational actors who carry out these processes. Finally, it identifies the information and knowledge artifacts manipulated by the concerned processes and organizational actors.

3.4.2 The Understanding stage

The knowledge gathered during the first stage is used during the Understanding stage to build shared visions of the organizational solution and the corresponding software problem. These shared visions combine articulated knowledge and informative prototypes. The Understanding stage rests on the business processes and functional (Information System) architecture layers of the software solution architecture multi-layered model. The conceptual and organizational business processes architecture models provided by the business processes architecture layer contribute to building a shared vision of the organizational solution. Not only the business process layer provides a view of the organizational processes related to the software problem to be solved and supported by the existing software solution (if such a solution exists). But also, it takes into account the impacts of strategic decisions on business processes. Such impacts refer to the characteristics of the organizational solution. The information system architecture models expressed in terms of business functions and business entities contribute to building a shared vision of the software problem corresponding to the organizational solution. Such models, provided by the functional architecture layer, reflect the strategic decisions impacts on the architecture of the organization's information system. These impacts originate either directly from the strategic layer or indirectly from interactions between the business processes and the functional

architecture layers. To avoid knowledge asymmetries, domain ontologies which provides organizations with a common understanding of business domains main concepts may be used, updated or built from scratch. In particular, domain ontologies provide knowledge foundations of repositories of the business entities and the business functions manipulated by the organizational processes. Such repositories play a critical role in building shared visions of the organizational solution and the software problem.

3.4.3 The Construction stage

During the Construction stage, the stakeholders work together by combining and transforming their knowledge in order to define a shared vision of what the future system should be. The stakeholders generally build a sequence of informative prototypes in order to elicit and conciliate their points of view. Informative prototyping plays a knowledge engineering-related role during the Construction stage. On the one hand, it reduces uncertainty through integration of tacit knowledge owned by stakeholders into software artifacts called informative prototypes which catalyse many aspects of the shared vision of the future software solution. On the other hand, it contributes to knowledge creation since informative prototypes facilitate communication and interaction between stakeholders. Knowledge generated by this way may improve the shared vision of the future software solution. Such a common vision is embodied in a set of operational prototypes which make up a version of the required software solution to be used and evaluated by users during the Assessment stage. The construction of informative and operational prototypes relies on the organizational processes and functional architecture layers which permit identification of operational tasks supported by the future software solution and the business entities and business functions manipulated by these tasks. The description of the applicative and software architectures of the future software system is beyond the scope of this paper. More detailed information related to this topic is provided by (Dakhli 2008).

3.4.4 The Assessment stage

During the Assessment stage, the operational prototype issued from an iteration is evaluated by the organizational actors it is intended for. These actors use the operational prototype as support to their operational tasks within the organization. The Assessment stage results in new problems and requirements to be taken into account during the next iterations. Solutions proposed to these problems generally generate knowledge which update the shared vision of the software solution and then make this solution richer.

4 CONCLUSION AND FUTURE RESERACH DIRECTIONS

The framework presented in this paper has been used in a French Insurance company to develop a software system aimed at supporting the management of the customer's claims. Let us note that many architecture guides exist in this company but the software project teams do not always apply the rules they define. Furthermore, the iterative development approaches including software prototyping are not applied in this company where the software development process is based on the waterfall sequential model. Finally, knowledge management is still considered as a long term project and no resources has been allocated to this project until this year. The goal of the use of our framework within this company was to demonstrate to the strategic managers that software solutions in knowledge-intensive companies must be built according to three principles. Firstly, the software development process in knowledge-intensive organizations includes software engineering traditional activities (design, coding, testing,…) which are intertwining with knowledge engineering activities. Secondly, the waterfall software development process is not appropriate to build knowledge-intensive software systems. Moreover, the development of such systems requires iterative approaches like software prototyping or agile methods. Finally, modelling organizational processes architecture and functional (Information System) architecture is required to build effective knowledge-intensive software systems. The Discovery stage starts by sending a Request for Information to all the entities of this company located in Europe and in the rest of the world. No significant answers result from this Request for Information.

Then the project manager launched many workshops in order to gather knowledge about the organizational solution and the solution problem. During this stage the customers claims management process has been modelled at the conceptual and organizational levels. Moreover, the business entities and business functions manipulated by the customers claims management process are identified. Two main problems have been encountered while accomplishing this task. The first problem stems from the fact that the difference between a task and a business function is not easy to understand while the second problem is related to the interpretation asymmetries of the business functions and business entities identified. These problems were solved by organising workshops with a facilitator who doesn't belong to the company. The construction of informative prototypes was not easy since there is no prototyping culture within this company. During the Construction stage, the same problems have been encountered. Nevertheless, this stage was less difficult than the Discovery and Understanding stages since the process of solving such problems was more mastered then during the first two stages.

The application of the proposed framework to a real project results in many recommendations and future research directions. Firstly, the construction of domain ontologies is required to deal with misunderstanding, misinterpretations and knowledge asymmetries related notably to the activity, task, business function, and business entity identification and use. Secondly, the relationship between software design and ontologies has to be described formally in order to build effective informative and operational prototypes. Finally, domain ontologies has to be integrated in the software architecture multi-layered model.

The main contribution of this paper consists in explaining the software crisis in terms of knowledge gap and in stressing the relationships between tacit knowledge owned by the organizational actors, organizational processes and information system architecture. By contributing to the reduction of the knowledge gap in software engineering, the proposed framework offers an alternative solution aimed at minimizing the software crisis impacts. We think that the integration of domain ontologies in this framework may improve it by clarifying the intertwining nature of software engineering and knowledge engineering activities and improving the knowledge gap reduction process.

References

Alavi, M., and Leidner, D.E. (2001). Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues. MIS Quarterly, 25 (1), 107-136

Alchian, A.A., and Demsetz, H.: (1972). Production, Information Costs and Economic Organization, American Economic Review, 62 (5), 777-795.

Baetjer H. Jr. (1998). Software as Capital: An Economic Perspective on Software Engineering. The Institute of Electrical and Electronics Engineers, Inc., Piscataway, New Jersey.

Blackler, F. (1995). Knowledge, Knowledge Work and Organisations: An Overview and Interpretation, Organisation Studies, 16 (6), 1021-1046.

Boehm, B.W. (2006). A View of 20th and 21th Century Software Engineering. In L. Osterweil, D. Rombach, and M.L. Soffa (Eds), Proceedings of the 28th International Conference on Software Engineering (ICSE'2006), pp. 12-29, ACM Press, New York,

Brooks, F.P Jr. (1987). No Silver Bullet-Essence and Accidents of Software Engineering. Computer, 20 (4), 10-19.

Brooks, F.P Jr. (1995). The Mythical Man Month: Essays on Software Engineering. Reading, M.A., Addison-Weslay.

Buckley, W. (2001). Mind and Brain: a Dynamic System Model. In Geyer, F., and Van Der Zouwen, J. (eds). Sociocybernetics: Complexity, Autopoiesis, and Observation of Social Systems, Knowledge Sharing Over Social Networking Systems.

Dakhli, S. (1998). Le Prototypage. Thèse de doctorat, Université de Paris-IX Dauphine, Paris, Mars.

Dakhli, S.B.D. (2008). The Solution Space Organisation: Linking Information Systems Architecture and Reuse. In the Proceedings of the ISD'2008 Conference, Paphos, Cyprus, August 25-27, 2008. Springer-Verlag.

Davis, A. (2005). Requirements Management. Dorset House.

Dieste, O., Juristo, N., and Shull, F. (2008). Understanding the Customer: What Do We Know about Requirements Elicitation?. IEEE Software, 25 (2), 11-13.

Glinz, M., and Wieringa, R. (2007). Stakeholders in Requirements Engineering. Guest Editor's Introduction, IEEE Software, 24 (2), 18-20.

Hirschheim, R., and Newman M. (1988). Information Systems and User Resistance: Theory and Practice. The Computer Journal, 31 (5), 398-407.

Ilavarasan, K.V. and Sharma, A.K. (2003). Is Software work routinized? Some empirical observations from Indian software industry. The Journal of Systems and Software, 66 (1), 1-6.

Jensen, M.C., and Meckling, W.H. (1976). Theory of the Firm: Managerial Behavior, Agency Costs and Ownership Structure. Journal of Financial Economics, 3 (4), 305-360.

Kautz, K. and Mcmaster, T. (1994). The failure to introduce systems development methods: A factor-based analysis. In Proceedings of the IFIP TC8 Working Conference on Diffusion, Transfer and Implementation of Information Technology (Levine, L. Ed.), p. 275, IFIP Transactions A-45, North-Holland, Amsterdam.

Leavitt, H.J., (ed.). (1963). The Social Science of Organizations, Four Perspectives. Prentice-Hall, Englewood Cliffs, New Jersey.

Neill, C.J., and Laplante, P.A. (2003). Requirements Engineering: The State of the Practice. IEEE Software, 20 (6),40-45.

Newell, S., Robertson, M., Scarborough, H., and Swan, J. (2002). Managing Knowledge Work, New York, Palgrave.

Nonaka, I. (1994). A Dynamic Theory of Organisational Knowledge Creation. Organisation Science, 5 (1).

Nonaka, I., and Takeuchi, H. (1995). The Knowledge-Creating Company. Oxford University Press, New York.

Polanyi, M. (1967). The Tacit Dimension. London: Routledge.

Regnell, B., and Brinkkemper, J. Market-Driven Requirements Engineering for Software Products. A. Aurum and C. Wohlin (Eds), Springer-Verlag, 287-308.

Simon, H.A. (1983). Models of Bounded Rationality. (2 volumes), MIT Press, Cambridge.

Toffolon, C. (1996). L'Incidence du Prototypage dans une Démarche d'Informatisation, Thèse de doctorat, Université de Paris-IX Dauphine, Paris, Décembre.

Toffolon, C. (1999). The Software Dimensions Theory. In Joaquim Filipe (Ed.), Enterprise Information Systems, Selected Papers Book, , KLUWER ACADEMIC PUBLISHERS

Toffolon, C., and Dakhli, S. (2007). KNOC: A Knowledge-Oriented Cooperative Software Development Process. In the Proceedings of the ISD'2008 Conference, National University of Ireland, Galway, August 29-31, 2008. Springer-Verlag.

Turner, J.R. and Müller, R. (2003). On the Nature of the Project As a temporary Organization. International Journal of Project Management, 21, 1-8.

Wiegers, K.E. (2003). Software Requirements. 2nd Edition, Microsoft Press.

Zack, M.H. (1999). Managing Codified Knowledge. Sloan Management Review, 40 (4).