A Knowledge Management Approach for Software Engineering Projects Development

Paulo Carreteiro[1], José Braga de Vasconcelos[1], Alexandre Barão[1], Álvaro Rocha[2]

[1] Knowledge Management and Software Engineering Research Group
Universidade Atlântica
Fábrica da Pólvora de Barcarena, 2730-036 Barcarena, Portugal
{pcarreteiro, jose.braga.vasconcelos, abarao}@uatlantica.pt

[2] Department of Informatics Engineering
Universidade de Coimbra
Pólo II - Pinhal de Marrocos, 3030-290 Coimbra, Portugal
amrocha@dei.uc.pt

Abstract. Aiming to emphasize the effect of knowledge management practices during software development projects, this research paper presents a first approach to cope with knowledge management and engineering practices across software development projects. The main goal is to define a roadmap for representative software development life cycle tasks during a typical software project development. The research introduces an ongoing architectural case study using software maintenance tasks as a means to enhance the knowledge flows within the organization. Software maintainers validate, correct and update knowledge from previous phases of software development life cycle through the application of back flushing technique at the software data warehouse. Further research developments will present a detailed guidance model for both research areas: knowledge management for software engineering combining insights across corporate software projects as a means of evaluating the effects on people and organization, technology, workflows and processes.

Keywords: Software Engineering; Knowledge Management; Knowledge Life Cycle; Software Maintenance; Software Development Life Cycle; KM Processes.

1 Introduction

In an organization individuals create documents, deal with software legacy applications, emails and many other tools where their everyday work is documented. Additionally, there's an informal or formal exchange of ideas with their peers to clarify doubts or discuss certain topics. All of these are contents for the three levels of refinement to knowledge, which are: data; information; and knowledge [1], [2].
Organizational knowledge resides in individuals or in a group, it can be explicit or tacit, regarding if it is documented or in the minds of individuals and having as scope the way it is applied, who accesses it and what activities it supports. Explicit knowledge can easily be shared, accessed and used. On the other hand, tacit

Á. Rocha et al. (eds.), *New Advances in Information Systems and Technologies*,
Advances in Intelligent Systems and Computing 444,
DOI 10.1007/978-3-319-31232-3_6

knowledge cannot, and it is acquired along the years by individuals, leaving the organization at the end of the day.
Knowledge Management (KM) aims at the individuals, their knowledge and the flows between them. Knowledge life cycle, defines the phases of organizational knowledge [2]. These phases are: (1) *Creation*, the origin of the cycle, occurs by informal or formal exchange of ideas and information either from internal or external sources; (2) *Capturing*, follows creation, it is the moment when it becomes explicit/documented; and (3) *Transforming* knowledge, which means organizing, mapping and converting it into an interpreted form. After it has been documented and organized, it can be *deployed*, meaning it can be accessed. Accessibility does not mean it reaches individuals, therefore *sharing* is when knowledge is used. The last phase is the goal of KM, which is the *usage*, allowing individuals to create more knowledge and therefore closing the cycle.

Although every project is unique and each model for software development has its specific methodology, all of them bare in common a complex set of tasks to achieve the final goal. According to the SWEBOK[1], *Software Engineering* (SE) is "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software". The aim of SE is the improvement organization productivity and quality by applying technologies and project management practices in activities related to the software development. Individuals that work in software development projects, also known as "knowledge workers" [1] must be able to interpret the sponsor's needs and transform them into coded language, thus SE is a discipline where one has to master both social and technical skills [4].

There are several models like cascade, agile, v-model, spiral, etc. for the *Software Development Life Cycle* (SDLC), these are approaches that determine the tasks and deliverables for software creation. In general, they are composed by six phases, allowing knowledge workers to specify and change software requirements into the final product or deliverable [5]. *Requirements specification* and *Analysis,* to interpret and explicit the client's needs; *design,* to transform requirements (business knowledge) into a plan (technical knowledge); *coding,* where the plan is implemented; *testing,* a proof of concept that the software meets the requirements; software *deployment,* when it is shared and implemented and finally *maintenance,* to support the software usage. Sponsors, who have the business know-how, interact with the knowledge workers to whom they transfer the aforesaid business knowledge, so that they can design models to transfer it into technical knowledge, meaning models that describe the software [6]. Combining distinct approaches, such as KM and SE, the key challenge of our research is to answer the following question: "What kind of KM framework is needed to help Software Maintenance knowledge workers?". In section 2, we present an overview of KM for SE, identifying the challenges organizations face to apply knowledge management practices. With the aim of answering our research question, in section 3, the MIMIR Framework is presented, namely: (A) The MIMIR Model Overview; and (B) The MIMIR Methodology.

[1] Software Engineering Body Of Knowledge

Finally, in section 4, we present our preliminary conclusion regarding our research's key challenge.

2 Knowledge Management for Software Engineering

Dependency on technology grows each day, increasing the costs for developing software, making it a larger piece of the "companies cake" called budget as well as boosting the amount and diversity of information that organizations must deal with [2], [7]. The ability to become leaner without losing efficiency and quality are challenges that these knowledge intensive companies must overcome and the trigger for a KM approach to the SDLC. Some of the challenges that organizations must deal with are presented next, as well as a KM approach regarding the SDLC phases. Knowledge is not a recipe, it is a set of procedures for dealing with a concrete situation, a routine. Knowledge should allow organizations to cope with different situations, anticipate implications and assess its effects [3].

2.1 Organizational Challenges

There is a significant amount of documents and artifacts produced during the phases of the SDLC, recording knowledge gathered along each sprint or project. Dealing with their usage is a challenge for the organizations. How to access it, along with ways to research what is really needed in a particular context, makes all this documentation unusable and difficult to share. Moreover these knowledge so dispersed that makes it hard to update [2], [7]. While some knowledge is recorded, some still remains in the individuals' mind [7], meaning that at the end of the day knowledge leaves the organization. Heuristics that individuals attain over several years cannot be transmitted within months to others and when they leave the company a knowledge gap is create, sometimes with complex and serious consequences for the organization. To manage knowledge and it is flows is of the utmost relevance. However, KM is not a product that the company can acquire, KM must be implemented in time [6] and involves processes, technology but above all individuals and organizations to enable a sharing culture, where the first ones must participate and the second ones must encourage, KM = People + Process + Technology [8].

2.2 KM/SE Approach

The SDLC is a knowledge-intensive environment where KM processes fit like a glove. It can function as a complement to support individuals during the SDLC phases to enhance productivity and quality [7]. The map we present in fig. 1 evidences the proximity between the Knowledge Life Cycle processes and SDLC activities. SDLC activities transform tacit knowledge into explicit [9], and when properly documented, they are organizational knowledge reification activities.

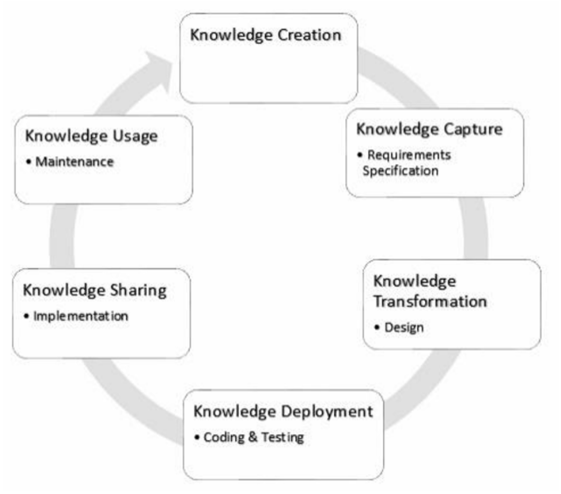

Fig. 1. Mapping KLC processes versus SDLC activities.

Knowledge in software development projects is diverse and growing in proportions. Organizations have problems identifying the contents, location and the best way to make use of it [2]. An important step towards a knowledge management project is the characterization of the knowledge assets. The domain knowledge includes business processes, decision-making, entrepreneurial, declarative and procedural knowledge, heuristics and informal knowledge [10]. For this paper, the activities of SDLC were segmented in three major process areas derived from fig. 1, regarding their outputs combined with their needs. They are Requirements Definition (RD), Software Development (SD) and Software Maintenance (SM), as seen in fig. 2. With this it is possible to identify all the documents created during the SDLC and determine within each one the more significant information.

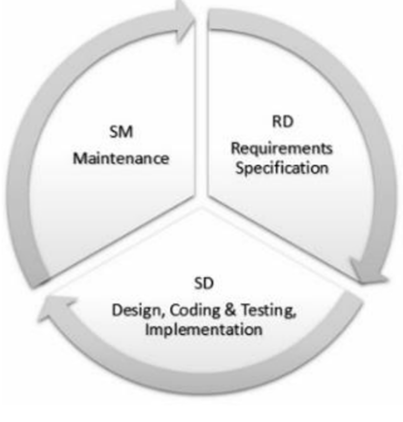
SM
Maintenance
RD
Requirements Specification
SD
Design, Coding & Testing, Implementation

Fig. 2. SDLC segmentation accordingly to their outputs and needs

Our main focus is the SM area, since in general KM processes have been less explored [2], [4], [5], [7], [10], [11]. It turns out that the documents produced in previous phases of SDLC are rarely used to aid in solving maintenance problems. Maintainers end up mainly resorting to read the legacy software code or to an informal exchange of ideas with their peers; 40% to 60% of the effort made in SM is spent understanding the software that is being maintained [12]. SM deals with real problems and its activity can be used to evaluate knowledge documented in the previous phases.

2.3 Research Methodology

Methodology is the study of methods, the set of techniques and processes used for research and development of a scientific work. Due to the nature of this research, of the various existing methodologies, the choice fell on the Design Science Research (DSR).
This is fundamentally a methodology for problem solving in order to create innovations that define the ideas, practices, technical capabilities or products that can be made more effective and efficient through analysis, design, implementation, management and the use of information technologies [16].

As well as the SDLC activities, which are widely discussed in this paper, so the DSR has its cycle of research, making it for this reason a methodology course used in conducting research related to information systems. There are seven guidelines in DSR aimed to assist researchers, reviewers, editors and readers to understand the requirements, namely: Design as an Artifact, Problem Relevance, Design Evaluation, Research Contributions, Research Rigor, Design as a Search Process and Communication of Research [16].

Design as an artifact aims at problem solving and problem knowledge, their understanding and resolution are solved with the creation and implementation of an artifact, hence our model overview in fig.3 and fig. 4. For an activity that is intended to be efficient and objective, as is the case of SM, it is a subject of *relevant importance* to the organization that up 60% of the time spent in maintenance is trying to understand the software itself. DSR is inherently iterative [16] therefore our *Design as Search Process* is based on literature revision, for reference to our work and with which we have a comparison table on section 3.3, but also provided from personal experience on a large software development organization.

DSR is essentially a search process to discover an effective solution to a problem [16], future work will continue to be developed for this research using the guidelines and the research life cycle from DSR.

3 MIMIR Framework

SM is a demanding and complex activity. Software maintainers need to master programming languages, the system architecture, understand data models, have procedural knowledge, software updates and their impacts and often this is done for several applications. Experienced individuals are chosen for these tasks, seniority and heuristics makes them the manager's top choice, thus highlighting that these are tasks that require a high level of organizational knowledge, sensibility and experience, due to their criticality and urgency. The MIMIR Model presented next, named after the Norse mythology god who guards the "Well of the Highest Wisdom", is an approach to a framework mapping information based on documents created in the SDLC phases to retrieve from them the utmost relevant knowledge for SM.

3.1 MIMIR Model Overview

This model was driven by the fact that maintainers deal with problems, therefore accessing relevant knowledge to aid them in the research is extremely important. Additionally, they can validate, correct or update the accessed knowledge, for which back flushing is applied, a technique used to correct data in the origin during the Extract, Transform and Load (ETL) process in Data Warehousing.

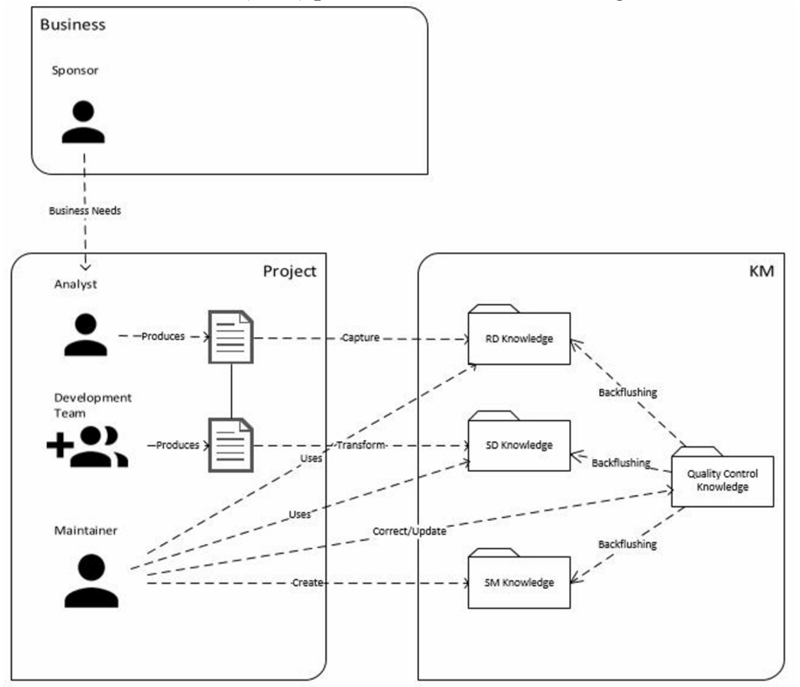

Fig. 3. MIMIR Model Overview

MIMIR Model performs a preliminary approach to the SDLC phases in organizations based in three segmentation areas as shown in fig. 2. Thus, as shown in fig. 3, several roles were defined as Unified Modeling Language (UML) stereotypes for participants, namely: *Sponsor*, *Analyst*, *Development Team* and *Maintainer*. Sponsor role is to define the needs for project and validator for the requirements quality, they can be external or internal to the organization. Analyst is the expert for the model, providing the requirements elicitation, application analysis and main source of knowledge in the project area. Development Team involves the individuals that participate in the project mainly in the coding phase. In smaller projects the Analyst can participate in the development team. Maintainer is the individual that provides support for the usage of the system, if the analyst has exclusive business knowledge he does not play the maintainer role, on other hand, if he is an Application Analyst he will surely be a Maintainer. *RD Knowledge* will retain all significant information regarding requirements elicitation. *SD Knowledge* will gather information from development phases of the project like technical plans or software design specification. *SM Knowledge* gathers maintainers' most frequently problem solutions, batch processing sequences, or application navigational sequences. *Quality Control Knowledge* is where the corrections and updates to previous knowledge remain until approval from experts like analysts to initiate the back flushing process.

3.2 MIMIR Requirements Capture Model

MIMIR is centered in the extraction of knowledge created in documents produced by the knowledge workers throughout the SDLC phases using the CMMI-DEV[2] model base. In this paper, in fig. 4 we only present the main concepts of the *RD Knowledge Base* feed in the requirements phase.

The elicitation of *Sponsor* needs from the *Analyst* will first produce the *Business Requirements Specifications* (BRS) document, a high-level business needs document and kick-off for the SDLC phases. The project ID goals and scope are important information that the *MIMIR-Extractor* module will acquire from the document. These will permit to create the project in the *MIMIR RD Database* as well as the first tags will be added to help in the search queries and the knowledge flow for the individuals, specially maintainers.

Also produced by the *Analyst* is the *Project Requirements Specifications* (PRS), in which the requisites for the development team will be described, each one with a specific ID, type and description and all referring to a functional area of the application or system. This information will be added to the RD Database project created previously. The requirement description will provide important knowledge in a natural language, easy and quick to interpret by the individuals specially *Maintainers*. Using the requirement ID as a key, MIMIR will map these descriptions to the list of artifacts that are affected in the project, registered later in the *SD Knowledge Database*.

The *Project Requirements Specifications Check-List* (PRS-CK) will close the requirements phase and consolidate the RD Database information from this document, and *MIMIR-Extractor* will retrieve information regarding the *Analyst* involved in the requirements elicitation, for previous expertise referral. The last step of the

[2] Capability Maturity Model Integration for Development – A model that provides guidance for improving organization's capability to develop quality products and services that meet the needs of customers and end users.

requirements phase is determined by requirements acceptance from the *Sponsor,* which will provide the trigger for the *Requirements Close and Dissemination* module (*MIMIR-RCD*) to make the knowledge from the *MIMIR RD Database* visible and distribute it to individuals in the *Development Team*.

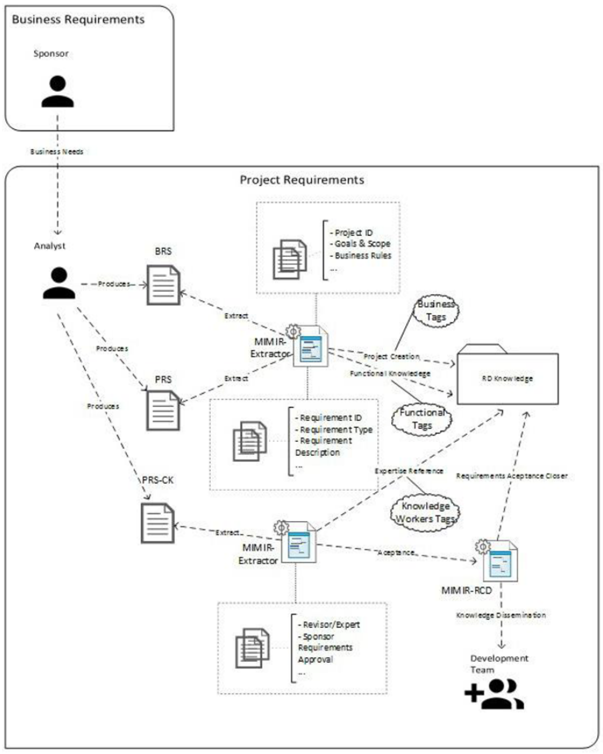

Fig. 4. MIMIR Requirements Capture Model

3.3 A Comparison with Related Work

We review other authors researches in KM for SE and especially through the SM approach as means of reference and guidance to our work, in which we have resumed in fig. 5 each one regarding their focus in the KM, SDLC and Quality dimensions.
For the P1 [13] the main focus is essentially on knowledge capture in the software requirements phase, enhancing the corporate memory as the main area of intervention, and it has left out the SM phase and the related problems. P2 [14] and P3 [15] have the main focus on the SM phase and its problems for the SDLC dimension, and for the KM dimension emphasize the knowledge share and the reuse of lessons learned. The previous phases of SDLC were left aside.
P4 [12] is the most in-depth research paper in this area, however, has no focus on quality control and knowledge correction. Although the primary focus is the SM

phase, it does not neglect the previous phases in the SDLC framework. Knowledge sharing has a minor attention and knowledge capture has a mild focus.
P4 is definitely a reference and guidance for the future of our research work because it has a well-defined ontology based on the maintenance problems to support other dimensions. On the other hand, P4 has reached the perception that the application of KM in SM phase promotes the creation of knowledge management culture (and approach) within the organization.

Dimensions	Projects	P1	P2	P3	P4	MIMIR
Knowledge Management	Knowledge Capture	●●●	❍	❍	●●	●●
	Knowledge Sharing	●	●●●	●●●	●	●●
	Knowledge Usage	●	●●●	●●●	●●●	●●●
Quality	Knowledge Quality Control & Correction	❍	❍	❍	❍	●●●
Software Development Life Cycle	Requirements Definitions	●●●	❍	❍	●●	●●●
	Software Development	●	❍	❍	●●	●●●
	Software Maintenance	❍	●●●	●●●	●●●	●●●

Legend:	❍	No Focus
	●	Little Focus
	●●	Mild Focus
	●●●	Large Focus

Fig. 5. Analogous Projects Review

4 Conclusion

Software development projects are knowledge-intensive and SDLC activities are the reification of the organizational knowledge. A lot of knowledge is documented but a lot still remains in the individuals' mind, therefore creating a potential (organisational) knowledge gap.
How to use documented knowledge and how to document the one that is not, are challenges for the organizations. SDLC proximity to KM can work as a facilitator to implement a KM project. However KM approaches to SE are mainly connected to the early phases of SDLC and especially in the requirements specification, leaving the software maintenance activities to a second plan. Up to 60% of the time spent in maintenance activities is re-acquiring knowledge and this is the main driver for our KM approach to SDLC, along with the fact that accessing knowledge maintainers can became validators for information created previously.
MIMIR approach applies documents provided from previous software engineering phases to capture knowledge from each software project and has a particular concern to use the maintainers as a means to validate and adjust the knowledge gradually inferred. The result is a KM software project charter combining a set of knowledge acquisition tasks across the SDLC framework.
As future work, regarding our research question, our aim is to present a detailed MIMIR model, methodology and tool, based on software development organization,

combining insights across corporate software projects as a means of evaluating effects on people and organization, technology, workflows and processes.

Acknowledgements

We acknowledge the financial support of the Iberian Association for Information Systems and Technologies (AISTI), which permitted the registration in the WorldCIST'16 (4th World Conference on Information Systems and Technologies), held at Recife, Brazil, 22 - 24 March 2016, and consequently this publication.

References

1. Davenport, T. H. (2010). Process Management for Knowledge Work. Handbook on Business Process Management 1. Springer
2. Rus, I., & Lindvall, M. (2002). Knowledge Management in Software Engineering. IEEE Software
3. Cascão, F. (2014). Gestão de Competências, do conhecimento e do talento. Lisboa: Edições Sílabo, Lda
4. Alawneh, A. A., Hattab, E., & Al-Ahmad, W. (2008). An Extended Knowledge Management Framework during the Software Development Life Cycle. International Technology Management Review
5. Bourque, P., & Fairley, R. E. (2014). SWEBOK v3.0 - Guide to the Software Engineering Body of Knowledge. New Jersey: IEEE Computer Society.
6. Camacho, J. J., Sanches-Torres, J. M., & Galvis-Lista, E. (2013). Understanding the Process of Knowledge Transfer in Software Engineering: A Systematic Literature Review
7. Natali, A. C., & Falbo, R. d. (2005). Knowledge Management in Software Engineering Environments. Vitoria
8. Leistner, F. (2010). Mastering Organizational Knowledge Flow. New Jersey: John Wiley & Sons, Inc
9. Aurum, A., & Ward, J. (2004). Knowledge Management in Software Engineering - Describing the Process. 2004 Australian Software Engineering Conference
10. Vasconcelos, J. B., Kimble, C., Miranda, H., & Henriques, V. (2009). A Knowledge-Engine Architecture for a Competence Management Information System
11. Isotani, S., Dermeval, D., Bittencourt, I., & Barbosa, E. (2015). Ontology Driven Software Engineering A Review of Challenges and Opportunities. IEEE Latin America Transactions
12. Anquetil, N., Oliveira, K. M., Sousa, K. D., & Dias, M. G. (2007). Software maintenance seen as a knowledge management issue. Information and Software Technology
13. Andrade, J., Ares, J., Garcia, R., Rodriguez, S., & Suarez, S. (2006). A Reference Model for Knowledge Management in Software Engineering.
14. Rodriguez, O. M., Vizcaino, A., Martinez, A. I., Piattini, M., & Favela, J. (s.d.). Applying Agents to Knowledge Management in Software Maintenance Organizations.
15. Talib, A. M., Abdullah, R., Atan, R., & Murad, M. A. (April de 2010). MASK-SM: Multi-Agent System Based Knowledge Management System to Support Knowledge Sharing of Software Maintenance Knowledge Environment. Computer and Information Science.
16. Hevner, A. R., Ram, S., March, S. T., & Park, J. (March de 2004). Design Science in Information Systems Research. MisQuartely, pp. 75-105.