Distributed Software Development with Knowledge Experience Packages

Pasquale Ardimento[1], Marta Cimitile[2], and Giuseppe Visaggio[1]

[1] University of Bari, Italy
{pasquale.ardimento,giuseppe.visaggio}@di.uniba.it
[2] Unitelma Sapienza University, Italy
marta.cimitile@unitelma.it

Abstract. In software production process, a lot of knowledge is created and remain silent. Therefore, it cannot be reused to improve the effectiveness and the efficiency of these processes. This problem is amplified in the case of a distributed production. In fact, distributed software development requires complex context specific knowledge regarding the particularities of different technologies, the potential of existing software, the needs and expectations of the users. This knowledge, which is gained during the project execution, is usually tacit and is completely lost by the company when the production is completed. Moreover, each time a new production unit is hired, despite the diversity of culture and capacity of people, it is necessary to standardize the working skills and methods of the different teams if the company wants to keep the quality level of processes and products. In this context, we used the concept of Knowledge Experience Package (KEP), already specified in previous works and the tool realized to support KEP approach. In this work, we have carried out an experiment in an industrial context in which we compared the software development supported by KEPs with the development achieved without it.

Keywords: Knowledge packaging, empirical investigation, distributed software development.

1 Introduction

Software development (production and maintenance) is a knowledge-intensive business involving many people working in different phases and activities [7]. The available resources are not increasing at the same pace as the needs; therefore, software organizations expect an increment in productivity [17]. However, the reality is that development teams do not benefit from existing experience and they repeat mistakes even though some individuals in the organization know how to avoid them. Considering that project team members acquire valuable individual experience with each project, the organization and individuals could gain much more if they were able to share this knowledge. Knowledge, therefore, is a critical factor and affects many different aspects of software development such as:

Y.T. Demey and H. Panetto (Eds.): OTM 2013 Workshops, LNCS 8186, pp. 263–273, 2013.

Accessing domain knowledge. Software development requires access to knowledge, not only about its domain and new technologies but also about the domain for which software is being developed. An organization must acquire new domain knowledge either by training or by hiring knowledgeable employees and spreading it throughout the team.
Acquiring knowledge about new technologies. It is difficult for developers to become proficient with a new technology and managers to understand its impact and estimate a projects cost when using it. When developers or project managers use a technology that a projects team members are unfamiliar with, engineers often resort to the learning by doing approach, which can result in serious delays. So, organizations must quickly acquire knowledge about new technologies and master them. Research produces knowledge that should be transferred to production processes as innovation in order to be valuable. Consequently, domain knowledge must be enriched by technical and economical knowledge that allows identifying the best approach for introducing new knowledge in processes together with the resources, risks and mitigation actions [16].
Sharing knowledge about local policies and practices. Every organization has its own policies, practices, and culture, which are not only technical but also managerial and administrative. New developers in an organization need knowledge about the existing software assets and local programming conventions. Experienced developers often disseminate it to inexperienced developers through ad hoc informal meetings; consequently, not everyone has access to the knowledge they need. Passing knowledge informally is an important aspect of a knowledge-sharing culture that should be encouraged. Nonetheless, formal knowledge capturing and sharing ensures that all employees can access it. So, organizations must formalize knowledge sharing while continuing informal knowledge sharing.
Capturing knowledge and knowing who knows what. Software organizations depend heavily on knowledgeable employees because they are key to the projects success [11]. However, access to these people can be difficult. Software developers apply just as much effort and attention determining whom to contact in an organization as they do getting the job done. These knowledgeable people are also very mobile. When a person with critical knowledge suddenly leaves an organization, it creates severe knowledge gaps. Knowing what employees know is necessary for organizations to create a strategy for preventing valuable knowledge from disappearing. Knowing who has what knowledge is also a requirement for efficiently staffing projects, identifying training needs, and matching employees with training offers. So, until knowledge is transferable or reusable, it cannot be considered as part of an organizations assets [11].
All these needs require both formalizing knowledge so that it is comprehensible and reusable by others that are not the author of the knowledge and experience packaging able to guide the user in applying the knowledge in a context. Given these premises, this paper describes an approach for Knowledge Packaging and Representation and reports results of an experimentation of the approach in an industrial context. In our proposed approach we have formalized a KEP and we have defined some packages that are stored in a Knowledge Experience Base

(KEB) [18,14,5,19]. The KEPs have a predefined structure in order to facilitate stakeholders in the comprehension and the acquisition of the knowledge that they contain. We have conducted a validation through a controlled experiment with the aim to answer to the following Research Question (RQ):

Does the proposed knowledge description approach increase the productivity of software development compared to the more traditional ones?
In order to answer this question we use the concept of *Productivity* considered as the effort spent to develop software. Due to the different size of applications considered, we normalized the *Productivity* using the *Function Point* factor. The rest of the paper is organized as follows: related works are described in Section 2; Section 3 illustrates, briefly, the proposed approach for knowledge representation, Section 4 illustrates the experiment planning and measurement model used; results of the study and lessons learned are presented in Section 5; finally in Section 6 conclusions are drawn.

2 Related Work

The topic of knowledge acquisition and collection is being studied by several research and industrial organizations [9,7,12,2,18]. This shows how the concept of KEB is quite mature, and several implementations [12,14,19] are performed. However, the knowledge structure in a KEB is not clear and common rules supporting knowledge sharing and acquisition are missed [15]. For example there are several KEB having a wider scope respect the generality of the collected knowledge [13]. This topic is particularly critical in distributed software development domain [6], were the experience transferring became an important value. In [10], authors focus on product design and development based on design KEB as a new attempt on the organic integration of knowledge management and product development. Moreover, [20] presents a notion of knowledge flow and the related management mechanism for realizing an ordered knowledge sharing and cognitive cooperation in a geographically distributed team software development process. Our approach aims to introduce the concept of KEP as software development KEB content structure. It makes it easier to achieve knowledge transfer among different stakeholders involved during software development and encourage the interested communities to develop around it and exchange knowledge.

3 Proposed Approach

3.1 Knowledge Experience Package Structure

Authors use the term KEP to refer to an organized set of knowledge contents and training units on the use of the demonstration prototypes or tools and all other information that may strengthen the packages ability to achieve the proposed goal. The KEP can be made up of all the elements shown in figure 1. The main component is the *Art&Practices* (*AP*). It describes the kind of knowledge contained in the package. *Evidence* component shows some empirical validation of

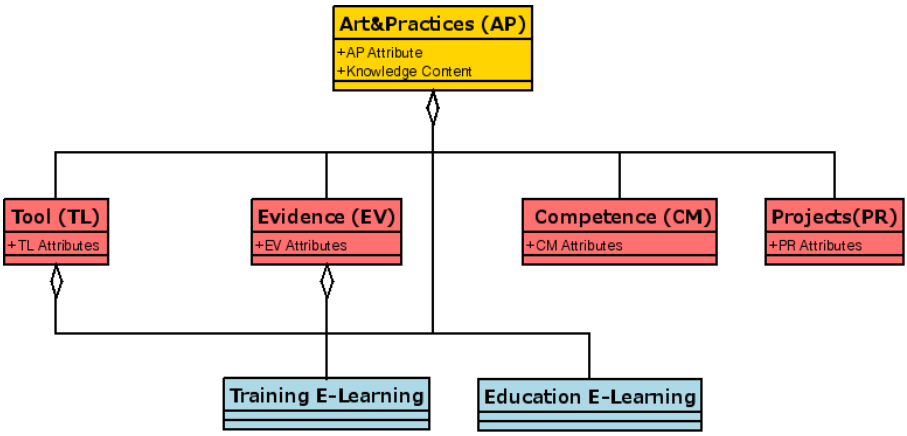


Fig. 1. Diagram of a Knowledge/Experience Package

the knowledge contained in AP. Similarly, the *Tool*, *Project* and *Competence* respectively describe the software tool, the project and all the competences useful to develop and use the described knowledge. A user can access one of the package components and then navigate along all the components of the same package according to her/his needs. The knowledge package must be usable independently of its authors and for this purpose the content must have a particular structure: distance education and training must be available through an e-learning system. In short, the proposed knowledge package contains knowledge content integrated with an *E-learning* function. Search inside the package starting from any of its components is facilitated by the component's *Attributes* and a more extended description of the knowledge is proposed in the *Content* part.

To facilitate the search, a set of selection classifiers and a set of descriptors summarizing the contents are used in *Attributes* component. The *Attributes* component contains the keywords and the problems the package is intended to solve, a brief description of the content and a history of the essential events occurring during the life cycle of the package, giving the reader an idea of how it has been applied, improved, and how mature it is. Also provides the following indicators: necessary competence to acquire it, prerequisite conditions for a correct execution of the package, adoption plan describing how to acquire the package and estimating the resources required for each activity. To assess the benefits of acquisition, they contain a list of: the economic impact generated by application of the package; the value for the stakeholders in the company that might be interested in acquiring the innovation. There are also indicators estimating the costs and risks. It is important to note that each kind of package has other specific attributes. Moreover, in the *Knowledge Content (KC)* a further description of the package is available. The *AP Knowledge Content* is the central one. It contains the knowledge package expressed in a hypermedia form in order to include figures, graphs, formulas and whatever else may help to understand the content. The *KC* is organized as a tree. Starting from the root

(level 0) descent to the lower levels (level 1, level 2, ...) is through pointers. At the increasing of the level of a node, the abstraction of the content decreases focusing more and more on operative elements. The root node is made up of a *Thoughtful Index* and one or more problems. The nodes are the answers to the problems, the solutions proposed for each of the announced problems. Each answer consists of the following: analysis of how far the results on which the innovation should be built can be integrated into the system; analysis of the methods for transferring them into the business processes; details on the indicators listed in the metadata of the *KC* inherent to the specific package, analyzing and generalizing the experimental data evinced from the evidence and associated projects; analysis of the results of any applications of the package in one or more projects, demonstrating the success of the application or any improvements required, made or in course; details on how to acquire the package. The research results integrated by a package may be contained within the same knowledge base or derive from other knowledge bases or other laboratories. The proposed KEP approach is supported by a tool called Prometheus. Prometheus [4,3] also allows to captures, share and retain content to ensure that one or more software processes are automated and managed. To achieve these results PROMETHEUS was interfaced with Drupal (https://drupal.org/). Drupal is a well known open source content management platform, supporting content capture, sharing and retention. The interested reader can find the tool and some package example at the following link [8] and further details are available in [1].

4 Experiment Planning

In the way to answer to the Research Goal (RG) introduced in the first paragraph, we propose an experiment consisting to compare the Prometheus based approach with a traditional one. The experimentation was conduct in collaboration with an PugliaIT Enterprise. We selected a set of projects and developers adopted Prometheus for their development or maintenance. We compared the obtained results with the results obtained by other developers in similar projects without Prometheus support. According to this we can better define our RQ in the following way: Analyze software production following the Prometheus approach with the aim of evaluating it with respect to *Productivity* following without use Prometheus from the view point of the software engineer in the context of company-projects.

4.1 Variables Selection

The dependent variable of the study is *Productivity*. *Productivity* indicates effort spent in man hours to develop a software application normalized on *Function Points*. The independent variables are the two treatments: the applications developed (only production) without Prometheus approach, software development (both production and maintenance) using Prometheus approach. Nine projects were considered, four already developed, and therefore without using Prometheus, and five developed using Prometheus. The projects refer to different

application domains with different characteristics such as number of developers, development software methodology and so on. Empirical investigation concerns both development of new applications and maintenance of already existing applications.

4.2 Selection of Experimental Subjects

The experimental subjects involved in the experimentation are developers all belonging to the same company. The developers of the four projects already developed did not know anything about Prometheus approach. All the developers of the five projects developed or maintained using Prometheus attended a training course on using Prometheus before to work on projects.

4.3 Experiment Operation

The experiment was organized in the following five experimental runs:

RUN_1 : retrospective analysis of four projects (P_1, P_2, P_3 and P_4) already developed (legacy projects) for each of one was available documentation;
RUN_2 : five projects (P_5, P_6, P_7, P_8 and P_9) developed from scratch;
RUN_3 : maintenance on P_5, P_6, P_7, P_8 and P_9;
RUN_4 : maintenance on P_5, P_6, P_7, P_8 and P_9;
RUN_5 : maintenance on P_5, P_6, P_7, P_8 and P_9.

RUN_2, RUN_3, RUN_4 and RUN_5 refer to the same projects. All runs from the third to the fifth only refer to adaptive maintenance. Four data collections were carried out every fifteen days for each experimental run from second to the fifth.

At the beginning of each run from 2 to 5, each experimental subject received username and password to login Prometheus platform in order to try, access and possibly evolve KEPs stored in it. Researchers measured Function Points for all nine projects using theory of *Function Points analysis* promoted and defined by The International Function Point Users Group (IFPUG) (http://www.ifpug.org/). During RUN_1, data have been collected from both available documentation and through interviews to the project managers. Collected data of all runs were stored on spreadsheets and, finally, the tool Statistica 7.0 developed by StatSoft (http://www.statsoft.com/) was used to execute all the analyses.

4.4 Projects Characterization

All nine industrial projects analyzed in our research present the features of enterprise applications, such as: an extended purpose; a high number of users, a required performance adequate to the application usage. Table 1 shows information about the all the projects. Several project were developed using a *Water-Fall (WF)* software development methodology. It is important to note that number of developers is always different from one project to other one.

It is important to note that the software methodologies used are only those ones known by the company involved in experimentation.

Table 2 shows how each development team was assigned to each experimental run. Assignment was made so that each team worked with a project, and as consequence for each methodology, not more than once.

The turnover of the developers involved in projects from P_5 to P_9 was continuous and occurred at the end of each experimental session through the reallocation of each team on a project different from the previous year. This turn-over was performed to assess the transferability of the use of the knowledge base regardless of the specific project for which you use. Finally, different development methodologies were used to assess the transferability of the use of the knowledge base regardless of the specific development methodology used. Finally, the various dimensions of the development team, as well as the different dimensions of software development, have been normalized using Function Points as explained in the following paragraph.

4.5 Measurement Model

Productivity is defined as:

$$Productivity = \frac{Size}{ManEff} \tag{1}$$

Size is the total number of Function Points (FP_A) added to the application to satisfy requirements of production or maintenance:

$$FP_A = FPd_A + FPr_A \tag{2}$$

FP_A is obtained summing FPd_A, that is the number of the Function Points developed ex-novo, and FPr_A, that is the number of Function Points already developed and reused thanks to Prometheus. ManEffort is the time, expressed

Table 1. Projects description

Id.	Brief Description	Software methodology	Developers
P_1	Registration and testing of autovehicle data history	WF	3
P_2	Electronic management of currency exchanges	WF	6
P_3	Management of bank stocks online	WF	5
P_4	Automation of human resources management	WF	2
P_5	Management of utilities energetic	WF	5
P_6	Management of Curriculum Vitae of a company	POD	5
P_7	Sharing of tacit and non structured knowledge present in an industrial	eXtreme Programming	5
P_8	Support the migration to a new SAP version or maintenance of a SAP installation	Light Weight Rational Unified Process	5
P_9	Upgrading of a SAP preconfigured system	LASAP	5

Table 2. Assignment of a development team to Project

Id.	RUN_2	RUN_3	RUN_4	RUN_5
T1	P_5	P_6	P_7	P_8
T2	P_6	P_7	P_8	P_9
T3	P_7	P_8	P_9	P_5
T4	P_8	P_9	P_5	P_6
T5	P_9	P_5	P_6	P_7

in man days (1 man day = 8 hours), spent for the development of a new functionality of an application. It is:

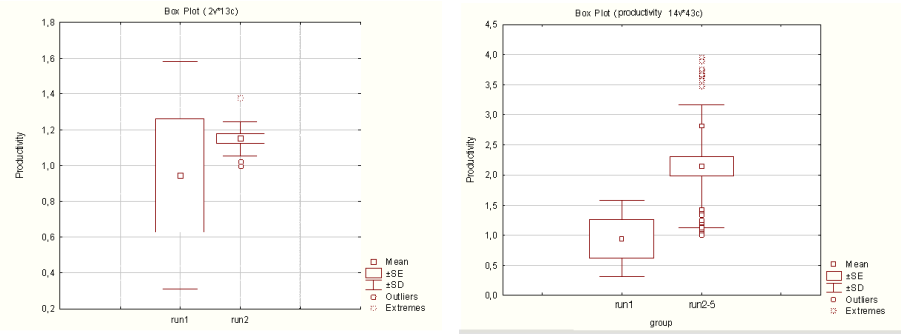
$$ManEff_A = ManEffd_A + ManEffr_A \tag{3}$$

where $ManEff_A$ is the total time for developing all Function Points added to application. It is obtained summing $ManEffd_A$, that is the time spent for developing ex-novo Function Points, and FPr_A, that is the time spent to integrate Function Points already developed and reused thanks to Prometheus.

5 Experimental Results

5.1 Productivity: RUN1 against RUN2

A first evaluation of the obtained result was made comparing the *Productivity* evaluated in RUN_1 with *Productivity* of RUN_2. In RUN_1, without Prometheus, the *Productivity* range is wider than in RUN_2 where developers, were supported by Prometheus. The results are shown in figure 2(a), where the box plots of *Productivity* evaluated during RUN_1 and RUN_2 are reported. From the figure we can observe that in RUN_1 *Productivity* range goes from 0,11 FP per day to



(a) RUN_1 and in RUN_2 (b) RUN_1 and in $RUN_{2,3,4,5}$

Fig. 2. Productivity with Prometheus and without Prometheus

1,58 FP per day with a mean value that is 0,94. In RUN_2, instead, *Productivity* range goes, except for outliers values, from 1,00 FP to 1,37 FP per day, and the mean value is 1,15 FP per day. Also, the dispersion of the results is very high for RUN_1 differently from RUN_2. Consequently interesting questions could be the following ones: Why is the range of FP much higher in RUN_1 than in RUN_2? Why is the mean value higher in RUN_2 than in RUN_1? In our opinion usage of Prometheus has made more predictable number of FP developed per day probably because of the systematic and process automation obtained by the use of CMS interfaced with Prometheus. It seems as if the performances are independent from the used technique.

5.2 Productivity: RUN_1 against $RUN_{2,3,4,5}$

Successively we compare *Productivity* results of RUN_1 with *Productivity* deriving from $RUN_{2,3,4,5}$. *Productivity* goes from mean value of 0,94 FP per day in RUN_1 to the mean value of 2,14 FP per day using Prometheus in the other four experimental runs. In each experimental run Prometheus is always better than the traditional approach and this confirms our hypothesis that Prometheus can increase Productivity of FP per day. Always observing figure 2(b) it is possible to note that the value mean of *Productivity* predictability for RUN_3, RUN_4 and RUN_5 is lower that value mean of RUN_2 probably because of increasing reuse of package.

5.3 Reuse: RUN_3, RUN_4, RUN_5

Figure 3 shows the trend of reuse of packages during $RUN_3, 4, 5$. Reuse grows with the populating of Prometheus. It is important to specify that this is not always true because it depends on matching between requirements to be developed to KEPs present in Prometheus. In other terms reuse grows or could grow with the populating of Prometheus.

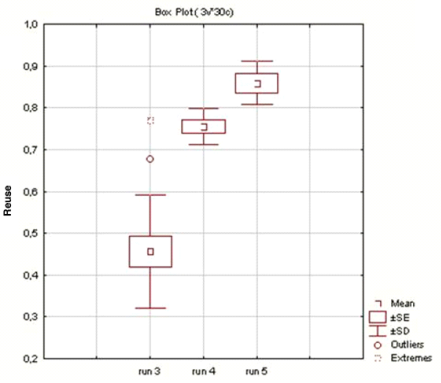


Fig. 3. Reuse in in $RUN_{3,4,5}$ (with Prometheus)

6 Conclusions

This paper proposes an approach based on the concept of KEP for knowledge transferring as further support for the software development. The proposed approach had previously been implemented through a KEB called Prometheus. To validate the approach, an empirical investigation was conducted. The experiment was carried out in an industrial context and consisted of a comparison between proposed approach and the traditional approaches used in the company in terms of *Productivity*. The collected results provide some lessons learned about Prometheus. Indeed Prometheus with respect to the traditional ones systematically formalizes and automates knowledge capturing, sharing and retention. Moreover it promotes reuse of developed functionalities reducing in this way the time necessary to produce them. It is clear that, in order to generalize the validity of the lessons learned proposed in this work, many replications, statistical validation and further studies, extended to other contexts, are needed. It is important to note that the selected set of experimental subjects, even if variegate, is not completely representative of the population of all software developers. As consequence, at this first stage, it is not possible to generalize the results of the empirical investigation. Rather, results represent a first important step towards this direction.

References

1. Ardimento, P., Baldassarre, M., Cimitile, M., Visaggio, G.: Empirical experimentation for validating the usability of knowledge packages in transferring innovations. In: Filipe, J., Shishkov, B., Helfert, M., Maciaszek, L. (eds.) ICSOFT/ENASE 2007. CCIS, vol. 22, pp. 357–370. Springer, Heidelberg (2009)
2. Ardimento, P., Boffoli, N., Cimitile, M., Persico, A., Tammaro, A.: Knowledge packaging supporting risk management in software processes. In: Proceedings of IASTED International Conference on Software Engineering SEA, Dallas, pp. 30–36 (2006)
3. Ardimento, P., Caivano, D., Cimitile, M., Visaggio, G.: Empirical investigation of the efficacy and efficiency of tools for transferring software engineering knowledge. Journal of Information & Knowledge Management 7(03), 197–207 (2008)
4. Ardimento, P., Cimitile, M.: An empirical study on software engineering knowledge/Experience packages. In: Jedlitschka, A., Salo, O. (eds.) PROFES 2008. LNCS, vol. 5089, pp. 289–303. Springer, Heidelberg (2008)
5. Basili, V., Caldiera, G., McGarry, F., Pajerski, R., Page, G., Waligora, S.: The software engineering laboratory: an operational software experience factory. In: Proceedings of the 14th International Conference on Software Engineering, ICSE 1992, pp. 370–381. ACM, New York (1992)
6. Boden, A., Avram, G., Bannon, L., Wulf, V.: Knowledge management in distributed software development teams - does culture matter? In: Fourth IEEE International Conference on Global Software Engineering, ICGSE 2009, pp. 18–27 (2009)

7. Gendreau, O., Robillard, P.N.: Knowledge acquisition activity in software development. In: Rocha, Á., Correia, A.M., Wilson, T., Stroetmann, K.A. (eds.) Advances in Information Systems and Technologies. AISC, vol. 206, pp. 1–10. Springer, Heidelberg (2013)
8. http://prometheus.serandp.com/en/content/iterative-reengineering-method-based-gradual-evolution-legacysystem
9. Jedlitschka, A.: An empirical model of software managers' information needs for software engineering technology selection: a framework to support experimentally-based software engineering technology selection. PhD thesis (2009)
10. Jin, H., Peng, W.: Study on product design and development based on design knowledge base. In: Proceedings of the 2009 Second International Symposium on Computational Intelligence and Design, ISCID 2009, vol. 1, pp. 463–466. IEEE Computer Society, Washington, DC (2009)
11. Foray, D., Kahin, B.: Advancing Knowledge and The Knowledge Economy. In: Advances in Intelligent Systems and Computing, vol. 206. MIT Press (2006)
12. Klein, A., Altuntas, O., Husser, T., Kessler, W.: Extracting investor sentiment from weblog texts: A knowledge-based approach. In: CEC, pp. 1–9 (2011)
13. Klein, M.: Combining and relating ontologies: An analysis of problems and solutions (2001)
14. Malone, T.W., Crowston, K., Herman, G.A.: Organizing Business Knowledge: The MIT Process Handbook. MIT Press, Cambridge (2003)
15. Qian, Y., Liang, J., Dang, C.: Knowledge structure, knowledge granulation and knowledge distance in a knowledge base. Int. J. Approx. Reasoning 50(1), 174–188 (2009)
16. Reifer, D.J.: Is the software engineering state of the practice getting closer to the state of the art? IEEE Softw. 20(6), 78–83 (2003)
17. Rus, I., Lindvall, M.: Knowledge management in software engineering. IEEE Software 19(3), 26–38 (2002)
18. Schneider, K., Schwinn, T.: Maturing experience base concepts at daimlerchrysler. In: Software Process: Improvement and Practice, pp. 85–96 (2001)
19. Schneider, K., von Hunnius, J.-P.: Effective experience repositories for software engineering. In: Proceedings of the 25th International Conference on Software Engineering, pp. 534–539 (2003)
20. Zhuge, H.: Knowledge flow management for distributed team software development. Knowledge-Based Systems 15(8), 465–471 (2002)