ELSEVIER

Information and Software Technology 42 (2000) 965–971

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.nl/locate/infsof

From process improvement to people improvement: enabling learning in software development

R. van Solingen[a,*], E. Berghout[b], R. Kusters[c], J. Trienekens[c]

[a]*Quality and Process Engineering Department, Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, D-67661 Kaiserslautern, Germany*

[b]*Department of Information Systems and Software Engineering, Delft University of Technology, P.O. Box 356, 2600 AJ, Delft, The Netherlands*

[c]*Department of Information and Technology, Faculty Technology Management, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

Abstract

The importance of people factors for the success of software development is commonly accepted, because the success of a software project is above all determined by having the right people on the right place at the right time. As software development is a knowledge intensive industry; the 'quality' of developers is primarily determined by their knowledge and skills. This paper presents a conceptual model of nine 'learning enablers' to facilitate learning in software projects. These enablers help identifying whether individual and/or organisational learning is facilitated. The main question addressed in this paper is: 'Which factors enable learning in software projects and to what extent?'

Keywords: Learning; Process improvement; Knowledge management; Software development; People

1. Introduction

In 1994, software measurement was introduced in the R and D department of Schlumberger RPS, by applying the Goal/Question/Metric (GQM) approach [6,19]. One of the RPS experiences was that the interpretation process of measurements was most important [12]. During the interpretation of measurement data, the project team analysed and evaluated their day-to-day processes. Those interpretations were done in structured meetings called 'feedback sessions'. During feedback sessions measurement data is interpreted, conclusions were drawn, and action points were defined.

It appeared that the most important driver for improvement in these industrial programmes was that the software developers were actually *learning* how to improve their activities. This led to the conclusion that establishing an explicit learning process in software development projects is the main challenge of today's industry. Consequently, a research was started to identify guidelines for further optimisation and enablers of learning. These learning enablers were identified through studying various theories about learning and have been validated in software improvement projects. In this paper an overview of the main findings is given. Summarised the following research question is addressed:

Which factors enable learning in software projects and to what extent?

The validation of the conceptual model is a complex and still an ongoing task. Therefore, this paper is focussed on the presentation of elements of the conceptual model. These elements are primarily based on seven case studies at Schlumberger RPS and in line with experiences at Bosch, Digital, Dräger, Ericsson, Tokheim, Proctor and Gamble and several other companies. However, this paper cannot include a full analysis of all case studies. Only, typical case-study conclusions are described in association with the learning enablers. For more details of the case studies we refer to Ref. [16].

In this paper, first, a brief introduction to learning theory is given. Second, a conceptual model is presented and the factors for enabling learning are presented. The learning factors are discussed in association with typical case-study results. The paper ends with overall conclusions.

2. Learning theory

In this section learning theory is investigated. The most

* Corresponding author.
E-mail addresses: solingen@iese.fhg.de (R. van Solingen), e.w.berghout@its.tudelft.nl (E. Berghout), r.j.kusters@tm.tue.nl (R. Kusters), j.j.m.trienekens@tm.tue.nl (J. Trienekens).

0950-5849/00/$ - see front matter
PII: S0950-5849(00)00148-8

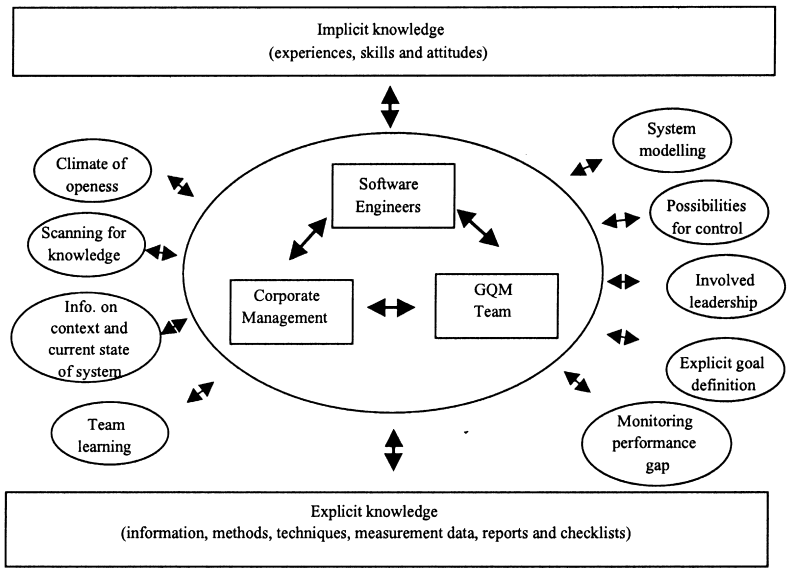

Fig. 1. Experiential Learning [10].

essential concepts are described. Additional information can be found in the particular references.

Learning is the process by which existing knowledge is enriched or new knowledge is created [23]. Learning deals with expanding *knowledge*. Knowledge is the personal ability that enables a person to perform a certain task [23]. This ability is the product of information (I), experience (E), skill (S) and attitude (A) of a person at a certain time ($K = I{\cdot}ESA$) [23]. Several classifications of the process of learning are described in the literature. For example: cognitive versus motor learning [5], declarative versus procedural learning [2], explicit versus implicit learning [21], or rationalistic versus empirical learning [23].

Nonaka and Takeuchi distinguish four learning *processes* [16]:

- 'socialising': a learning process between people in which *implicit* (tacit) knowledge is transferred by copying, imitating, master/pupil relationships, and experiencing by trial and error;
- 'externalising': a learning process, individual or between people, in which implicit knowledge is made explicit by for example model building, dialogues, and hypothesis formulation;
- 'combining': a learning process in which *explicit* knowledge from different sources is combined by for example: studying, analysing, reconfiguring, and integrating;
- 'internalising': an individual learning process in which explicit knowledge is made implicit through learning by doing, creating routines, and enlarging operational efficiencies.

Although, all four learning processes are present during, and relevant for software development, in this article we focus on the *explicit* learning processes: externalising and combining. With this decision in mind, this article continues with an exploration of learning theory. Individual learning will be considered first, followed by group learning.

2.1. Individual learning

During individual learning, the knowledge of one single person expands. Experiential Learning theory [10] defines an explicit learning process, in which *experiences* are *transformed* into knowledge, through model building and model testing.

Experiences are divided into concrete experiences: observations like seeing, feeling or hearing, and abstract conceptualisations: theories and models about observations and their relationships. Transformations are divided into reflective observations — analysing observations and developing new models and theories, and active experiments — testing models and theories in practice. According to Experiential Learning theory, neither the experience nor the transformation alone is a sufficient condition to achieve learning (Fig. 1).

Following the different classes of experience and transformation, four different modes of learning are distinguished. These modes are:

- 'divergent learning' during which observations are analysed;
- 'assimilative learning' during which models are built;
- 'convergent learning' during which models are tested in practice;
- 'accommodative learning' during which experiments are observed.

According to Kolb, the combination of these four modes of learning produces the best conditions for learning. The

combination requires the learning process to include: 'observing phenomena, analysing them, developing models and theories about them and testing these theories and models in practice' [10].

2.2. Group learning

When considering learning in software development projects, it is important to realise that work is performed in a practical environment. Software development is carried out within teams, projects, departments and companies; it always concerns a group of people. The development processes and improvement objectives are shared. The learning process, therefore, demands 'group learning'.

The term group learning indicates that a set of people, over a period of time, share the same learning goals and learning processes. In such a situation, knowledge has to be shared among organisational members and to contribute to the synergy of the organisation [9]. This is also often termed: 'organisational learning'. Organisational learning is defined as a *skilled process in which knowledge is created, acquired, and transferred, and through which behaviour is modified based on the new knowledge and insights* [7]. It is important to note that organisations cannot learn: the individual *people* can learn and learn together [23].

This definition reflects that learning happens when new insights arise. Such new insights are, however, not enough. 'Without accompanying changes in the way that work gets done, only the potential for improvement exists' [7]. George Huber states similarly that learning occurs when 'the potential behaviours are changed' [8]. Behaviour does not need to be changed for every situation, but the potential ways of working need to be expanded. So, effective learning results in altering (potential) behaviour. If behaviour is not changed, learning has apparently not occurred.

Argyris and Schön identify two modes of learning [3]:

- Single loop learning. This is learning in which the actor only learns within the confines of his or her theory in use. There is a focus on the operational level: based on detecting and correcting errors, competencies and routines.
- Double-loop learning. Double-loop learning starts when an event is diagnosed as incompatible with the actors' current theory in use. With double-loop learning current theory and models are altered through new insights.

In practice, most organisations are only focussed on single loop learning [4]. Optimisation is only done within the current way of working. This in itself is not wrong. Through repetitive experiences, organisations get skilled in their work, and create competitive advantages based on these skills. However, sometimes new approaches become available that an organisation has no experience with. In such cases, it might be better to switch to such a new approach. This is double-loop learning, which many organisations tend to see as a threat because it conflicts with existing and established habits.

It is also dangerous for an organisation to constantly adopt new ways of working, because all knowledge gained until then might immediately become outdated. 'The known can be in many situations be preferred over the unknown' [14]. A balance should be found in optimising current processes (single loop learning) and experimenting with new approaches to find out whether those are much better than existing ones (double-loop learning). So, learning theory promotes a parallel application of optimisation of current practices and experimentation with new ones.

The skills and capabilities of learning organisations are [17]:

- 'aspiration' — the capacity of individuals, teams, and eventually larger organisations to orient toward what they truly care about, and to change because they want to, not just because they need to;
- 'reflection and conversation' — the capacity to reflect on patterns of behaviour and assumptions deeply hidden in a persons behaviour, both individually and collectively;
- 'conceptualisation' — the capacity to see larger systems and forces at play and to construct public, testable ways of expressing these views.

According to Senge, there are three categories of learning skills. First, there is the motivation to learn and improve. This includes having time for learning, learning objectives, interest in learning, etc. Management commitment for learning tasks is also one of the aspects that falls under aspiration. Second, there is the willingness to discuss deep assumptions. This is what Argyris and Schön call 'double-loop learning'. Finally, there is conceptualisation, which corresponds with model building and testing of the Experiential Learning theory [10]. These three skills and capabilities for establishing learning need to be addressed in software development.

Learning theory supports that a learning method should include making the goals for learning explicit [7]. Defining these goals is difficult, however, in a business environment it makes sense to base them on business goals. These goals will be different for different organisations. Differences are for example: the market in which an organisation operates, the type of product that is delivered, the organisation of the development terms, or the country in which the products will be used. Learning practices should be directed to the goals of the organisation [7].

The final aspect of organisational learning relevant for this paper is based on a phenomenon called 'creative tension' [17]. This is the difference between the current reality and a desired future. The gap between the current reality and the desired future should not be too large, because the objectives of the people become too abstract and concrete actions towards improvement are not clearly visible. On the other hand, the gap between current reality

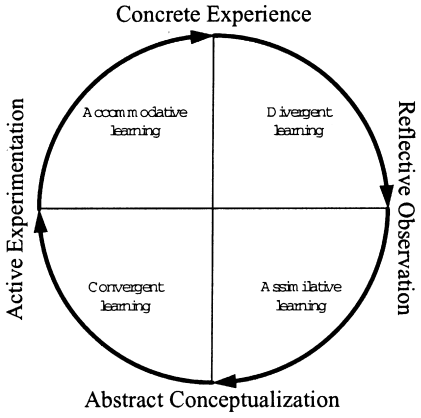

Fig. 2. Conceptual model of enabling factors for learning.

and the desired future should not be too small either, because this will result in no action at all, since the need for action might seem unnecessary. This creative tension principle indicates to set reachable objectives for learning.

3. Conceptual model of enabling factors

Based on above theory a conceptual model has been developed containing the most prominent enabling factors for learning in software development. This paper does not leave sufficient space to clarify the complete construction process of this conceptual model, which is mainly based on learning enablers described in Refs. [7,13,15,17]. For details on this construction process we refer to Ref. [20].

In Fig. 2, the conceptual model is depicted. In the centre the three main stakeholders are visualised with their interactions: software developers, management, and the learning support team (GQM team [19]). The learning process of these three groups of people is influenced by several learning enablers. The learning processes use and change both implicit and explicit knowledge. For more information on the interaction between these stakeholders and the influence on their learning processes we refer to Ref. [18].

The nine learning enablers will subsequently be described, together with what the enabler means within the context of software development.

3.1. Enabler 1: climate of openness

A climate of openness addresses the establishment of an environment in which free flow of information, open communication, sharing problems and lessons learned, and open debate of ways to solve problems, is available. Such a climate or 'learning culture' could seem a simple concept, however, turns out to be difficult to establish in practice. Research has indicated that current structures for control and management in organisations tend to disable such a climate of openness and this consequently decreases the commitment of developers [1,22]. The intrinsic motivation of developers is especially crucial for a creative and learning oriented environment. Practical actions that managers can take to increase the intrinsic motivation of developers are grouped in the following six categories [1]:

- 'Challenge', by matching the right developers with the right job assignments in such a way that employees definitely do not feel bored, but neither are overwhelmed or threatened by a loss of control.
- 'Freedom', by giving developers autonomy concerning the processes they apply. Management needs to set the goals, preferably as unambiguous as possible, however, the way to achieve these goals should be left to the developers themselves.
- 'Resources', by carefully allocating time and money. Time pressure can increase motivation as long as targets are perceived as attainable. Money should be assigned properly to prevent developers trying to find additional money themselves instead of doing their work.
- 'Work-group features', by carefully designing teams that are diverse, excited about their objectives, willing to support team-mates through more difficult periods, and where each member contributes a unique amount of knowledge.
- 'Supervisory encouragement', by praising creative efforts spent by their developers. Encouragement is not considered to be effective when given in extrinsic rewards such as financial bonuses. Freely and generously recognising creative work by employees already implies a significant encouragement. Managers should also not be sceptical towards new and rigorous ideas.
- 'Organisational support', by establishing sufficient organisational support for the developers in the organisation. This organisational support should enable learning efforts and supports learning processes. Furthermore, the value of learning should be emphasised by the procedures and systems in the organisation.

A climate of openness appears to be one of the most crucial prerequisites for organisational learning. It requires a context in which developers are willing to learn from their mistakes and willing to discuss underlying causes and models for these mistakes.

3.2. Enabler 2: scanning for knowledge

In the broadest sense this means that there should be a continuous search for knowledge that could be relevant or applicable in a particular learning situation. Scanning for knowledge from previous products, competitor products, similar products, or new methods is an important input to the requirements phase of a software project. The main point is that this loop does not build software product requirements every time from scratch but attempts to learn from previous experiences. Furthermore, knowledge can be collected from previous projects that created similar products. Carrying out ex post evaluations to find out

whether a certain used process model was adequate, is a good source of knowledge to increase learning effects. Double-loop learning also requires scanning for knowledge. Reading publications on achievements in software engineering by other software developers, is a way to scan knowledge. This way, experiences from other organisation can be assimilated into the own organisation. Having developers attend conferences, seminars and training is also a possibility.

3.3. Enabler 3: information on context and current state of the system

In order to learn, information is required on the current status of the system under development and the organisational context. A first step will be making processes explicit. A second step will be measuring the performance of these processes, or the current state of the product. Consequently, the effects of improvement actions can be measured.

For example, in one of the case studies a software process assessment was performed to make the status of current processes explicit. Knowing what the capabilities of an organisation are and making explicit which process actions they can use contributes to this learning enabler. If for example, process assessments indicate that configuration management is a weakness in the organisation, projects that have high product maintainability targets will know that they need to take some specific action.

3.4. Enabler 4: team learning

Team learning is an important part of an organisational learning process. It means that learning is established within groups that work together towards a shared vision and mutual objectives. Joint formulation of learning objectives, information sharing, discussion, and drawing conclusions together take place within team learning.

Team learning can be used to find out an appropriate way in which product requirements need to be specified to let the final product comply to them. It is also important that development teams learn the behaviour of different development processes. A specific process may not always give the same effect within different projects, for different products, with different team members. These differences and the causes for them should be determined. Measurement appears to be a powerful mechanism to enable group learning, particularly through the discussion on measurement results and through challenging each others' interpretations [19].

3.5. Enabler 5: modelling of the system under control

In order to control the software system under development, a model will be required from this system and its influencing factors. This can be done through process modelling, and modelling of the relationship between the product requirements and this process. Another example of useful modelling is the modelling of user groups and their mutual relationships to support the identification of all stakeholders [11].

In a case study, explicit models were made from the process that was intended to be used [20]. Also, models were made of the expected impacts of a particular process action. In one of the companies, for example, they introduced 'incremental development' by which the product was developed in three sequential increments, each expanding the previous one with specific functionality. The expectations of this change were modelled by formulating improvement hypotheses. The measurements showed indeed that these expectations were legitimate.

3.6. Enabler 6: possibilities for control

In order to steer a process towards the desired outcomes, possibilities for control should be available. This means that during a software project (corrective) actions can be taken whenever necessary. For example, when it appears that the intended product reliability level can not be reached, it should be possible to take action to improve that situation. In a double-loop learning fashion, the available set of process actions can be expanded with new ones that suit the specific organisation.

3.7. Enabler 7: involved leadership

The role of management is extremely delicate in enabling learning [1,7,17]. Insufficient attention from management gives learning an inadequate priority. The opposite of too much intervention from management normally leads to a learning project that is insufficiently supported by the developers (they feel over-observed or paternalised).

In a learning organisation, managers and the role of the manager differ from traditional production organisations. Senge [17] states the following differences: the manager is a designer of the learning organisation, a teacher regarding the view on reality, and a steward for the people they manage.

In the case studies it was observed that practical implementation of such a different management style is a difficult process, because both the manager and the developers were used to a different style. In an organisation where a manager always defines all detailed procedures that are to be used, and the manager suddenly leaves this freedom for the process to the developers, it is likely that developers also cannot cope with this freedom. Such a change in management style should therefore be carefully planned and a transition should be established. However, in creative intellectual work such management styles are often already present.

3.8. Enabler 8: explicit goal definition

In order to have clear targets towards learning, particular goals should be defined and made explicit. Learning

processes benefit if it is clear what the goals are and in which area learning is required to attain these goals. Both product and process goals should be stated explicitly. For example, regarding process goals, measurement goals are defined to monitor the performance of specific process actions, and the measurements are analysed explicitly to learn the effects of such a process action. For double-loop learning, explicit learning goals are defined to learn effects of process actions with which no experience exists. Expectations (hypotheses) should be explicitly specified regarding attainment of these learning goals, because expectations can be compared to actual values and reasons for differences can be identified.

An example of the use of explicit goals for learning was the identification of re-use effects in one of the case-study organisations [19]. The project team defined the explicit goal to measure the effects of software re-use on product reliability. Their expectation was that this contribution was high. The measurements showed indeed that the defect level of fully re-used modules was remarkably low. An important learning point from this project was the indirect effect of re-use on reliability. The project team learned that they were more strictly reviewing and testing re-used modules, because their confidence in these modules was initially lower, than the one they had developed themselves. The project team learned that both direct and indirect effects of re-use largely influence product reliability, and learned furthermore what these effects are.

3.9. Enabler 9: monitoring performance gap

Monitoring the differences between targeted and actual situation is an important prerequisite for learning. This monitoring both contains the identification of what is going well and what needs improvement. Through monitoring, developers get feedback on their way of working and learn where to improve. Again, monitoring a possible performance gap is not only done for both the product and development processes.

For example, in one of the case-study companies the following problem occurred. Due to the large number of countries being supplied and the large differences in government regulations across the countries, it was difficult to address all country specific requirements. This caused many ‘change requests’ after product release to the national representatives. A solution to this problem was to develop the country specific requirements in closer co-operation with the national representatives and, furthermore, use these requirements as input to the architecture of the product design. Consequently, as a result a product-architecture was designed that was capable of attaching country-specific customisations to the product after release.

4. Conclusions

As stated earlier in this paper, the importance and impact of people factors on the success of software development is commonly accepted. Knowledge and skills are an important input to a software development project, however, are also an important output, because they are continuously enhanced over time. To some extent developers always learn during software development projects and learning is also an important prerequisite to improve software development practices. Creating an organisational structure, in which effective learning is established, is a major challenge. We have illustrated that this is not a repeatable process for which only a procedure or manual needs to be written. On the contrary, managing the learning process of software developers appears to be a difficult and complex task, for which a specific management style is required. Establishing an eager and learning orientated environment is, essential. However, facilitating learning is not easy.

In order to do so, a conceptual model including nine ‘learning enablers’ is presented in this paper. This conceptual model is based on learning theory and based on experiences in various case studies. A first validation of this model has taken place [20]. The conceptual model could seem trivial in terms of elements. One could state that the enablers are vague and do not yet point out exactly ‘what’ needs to be done and ‘how’. Based on extensive literature research and work in practice, this is as far as we have come. We will continue our research on establishing learning organisations for software development.

This paper contains several points of attention for software managers in practice. By making the learning factors explicit, this paper hopefully contributes to improving the learning conditions in software organisations. We recommend that software development line managers and project managers consider these learning enablers continuously in their daily practice in order to increase the learning effectiveness of their developers. After all, it is those people that make the product.

References

[1] T.M. Amabile, How to kill creativity, Harvard Business Review September/October (1998) 77–87.
[2] J.R. Anderson, Cognitive Psychology and its Implications, 3rd ed., Freeman, New York, 1990.
[3] C. Argyris, D.A. Schön, Organizational Learning: a Theory of Action Perspective, Addison-Wesley, Reading, MA, 1978.
[4] C. Argyris, On Organizational Learning, Blackwell Publishers, Oxford, UK, 1992.
[5] K. Ayas, Design for Learning for Innovation, Eburon, Delft, 1997.
[6] V.R. Basili, D.M. Weiss, A methodology for collecting valid software engineering data, IEEE Transactions on Software Engineering SE-10 (6) (1984) 728–738.
[7] D.A. Garvin, Building a learning organisation, Harvard Business Review July/August (1993) 81–91.
[8] G.P. Huber, Organisational learning: the contributing processes and the literatures, Organization Science 2 (1) (1991) 88–115.
[9] M. Jelinek, Institutionalizing Innovation, Praeger, New York, 1979.
[10] D.A. Kolb, Experiential Learning, Prentice-Hall, Englewood Cliffs, NJ, 1984.

[11] R. Kusters, R. van Solingen, J. Trienekens, Identifying Embedded software quality: two approaches, Quality and Reliability Engineering International, Wiley, New York, 1999 (pp. 485–492).
[12] F. van Latum, M. Oivo, B. Hoisl, G. Ruhe, No improvement without feedback: experiences from goal oriented measurement at Schlumberger, Proceedings of the 5th European Workshop on Software Process Technology (EWSPT96), Nancy, France, Lecture Notes in Computer Science, vol. 1149, Springer, Berlin, 1996, pp. 167–182.
[13] A.C.J. de Leeuw, Organisations, Management, Analysis, Design and Change: a Systems Perspective, van Gorcum, Assen, The Netherlands, 1986 (in Dutch).
[14] J.G. March, Exploration and exploitation in organizational learning, Organization Science 2 (1) (1991) 71–87.
[15] E. Nevis, A. DiBella, J. Gould, Understanding organisations as learning systems, Sloan Management Review Winter pp. 73–85 (1995).
[16] I. Nonaka, H. Takeuchi, The Knowledge-Creating Company, Oxford University Press, New York, 1995.
[17] P.M. Senge, The Fifth Discipline: the Art and Practice of the Learning Organisation, Doubleday, New York, 1990.
[18] R. van Solingen, E.W. Berghout, E. Kooiman, Assessing feedback of measurement data: relating Schlumberger RPS practice to learning theory, Proceedings of the 4th International Software Metrics Symposium (Metrics'97), Albuquerque, 5–7 November, IEEE CS, Silver Spring, MD, 1997, pp. 152–164.
[19] R. van Solingen, E.W. Berghout, The Goal/Question/Metric Method: a Practical Guide for Quality Improvement of Software Development, McGraw-Hill, New York, 1999 (http://www.gqm.nl/; ISBN 0077095537).
[20] R. van Solingen, Product Focused Software Process Improvement: SPI in the Embedded Software Domain, BETA Research Series, no. 32, Downloadable from http://www.gqm.nl/, Eindhoven University of Technology, ISBN 90-386-0613-3, February, 2000.
[21] J. Swieringa, A.F.M. Wierdsma, On the way to a learning organisation: on learning and education in organisations (in Dutch), Wolters Noordhoff Management, 1990.
[22] D. Ulrich, Intellectual capital = competence × commitment, Sloan Management Review Winter (1998) 15–26.
[23] M. Weggeman, Knowledge Management (in Dutch), Scriptum Management, 1997.