Lecture Notes in Computer Science 2640
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Hong Kong
London
Milan
Paris
Tokyo

Scott Henninger Frank Maurer (Eds.)

Advances in Learning Software Organizations

4th International Workshop, LSO 2002
Chicago, IL, USA, August 6, 2002
Revised Papers

Springer

Volume Editors

Scott Henninger
University of Nebraska-Lincoln, Computer Science and Engineering
115 Ferguson Hall, CC 0115, Lincoln, NE 68588-0115, USA
E-mail: scotth@cse.unl.edu

Frank Maurer
University of Calgary, Department of Computer Science
2500 University Drive NW, Calgary, Alberta T2N 1N4, Canada
E-mail: maurer@cpsc.ucalgary.ca

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): D.2, K.6, H.5.2-3, I.2.4, K.3, K.4.3

ISSN 0302-9743
ISBN 3-540-20591-8 Springer-Verlag Berlin Heidelberg New York

Springer-Verlag is a part of Springer Science+Business Media

springeronline.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10969106 06/3142 5 4 3 2 1 0

Preface

The theme of the 4th International Workshop on Learning Software Organizations (LSO 2002) was "Balancing Agile Processes and Long-Term Learning in Software Organizations." The LSO Workshop series focuses on technical, organizational, and social solutions to problems of learning from past experiences and codifying the resulting best practices so they can be systematically used in subsequent software development efforts.

Through paper presentations, panels, and discussions, the workshop explored the issues of managing knowledge in dynamic domains requiring significant differences between organizations and between projects. Challenges discussed ranged from realistic assumptions on the added documentation burden LSO techniques may require to how effectively repositories have been used in the past to the team and social issues involved in applying solutions created by others. Experience-based approaches were discussed extensively and some reports of initial successes were given along with some instances where the experience base was underutilized.

Enabling organizational learning involves more than repositories, search engines, and training. At its core, it involves creating new work practices that value current practices while searching for improvements. The issues involved are both technical and behavioral, as effective technology may entice utilization, but experience has shown that other factors weigh in just as heavily.

There are currently no profound or final answers on these questions, nor are they expected for some time to come, if at all. Hence the need for continued research into these difficult issues. This workshop, and others to follow hope to begin to shed light on the issues so an effective and fruitful dialog can begin that can lead to significant contributions to the software engineering and knowledge management fields, amongst others.

The LSO workshop series has been designed as a communication forum that addresses the questions of organizational learning from a software point of view and builds upon existing work on knowledge management and organizational learning. It aims at bringing together researchers and practitioners for an open exchange of ideas and real-world experiences. Fostering interdisciplinary approaches is a key concern of this workshop series.

This year's workshop, held in conjunction with the XP/Agile Universe conference, emphasized the relationship between organizational learning and agile methodologies. The conference and workshop complemented each other and provided groundbreaking insights into current approaches used to establish learning software organizations.

We would like to take this opportunity to thank the many people who worked to help make this happen. Laurie Williams was instrumental in working with Frank Maurer to make workshop arrangements for the XP/AU conference. The program committee made the time and effort to provide valuable reviews of the papers. The authors should also be thanked for their willingness to revise and improve their papers into the forms found in this volume. Special thanks go to our keynote speaker, Günther Ruhe; his expertise spans the gap between LSO and decision support systems, and the full paper he submitted on

the topic is found in this volume. Thanks also go to Scott Ambler; his time and expertise provided valuable insights and promoted useful discussion in our "Can Agile Methods and Learning Software Organizations Support Each Other?" panel.

The LSO conference series continues to provide valuable discussions and exchanges of ideas which we are certain will have an impact on the future of software engineering techniques and practices. This year's conference was no exception, and it is expected that the discussions and contacts established at the workshop will pay dividends for years to come.

February 2003

Scott Henninger
Frank Maurer

Workshop Organization

Workshop Chairs

Scott Henninger, University of Nebraska-Lincoln (USA)
Frank Maurer, University of Calgary (Canada)

Program Committee

Klaus-Dieter Altoff, Fraunhofer IESE (Germany)
Giovanni Cantone, Università degli Studi di Roma (Italy)
Alexander Egyed, Teknowledge (USA)
Reidar Conradi, Norwegian Univ. of Science and Technology (Norway)
Raimund L. Feldmann, University of Kaiserslautern (Germany)
John Grundy, University of Auckland (New Zealand)
Mikael Lindvall, Fraunhofer, Maryland (USA)
Wolfgang Müller, University of Applied Science, Ludwigshafen (Germany)
Markus Nick, Fraunhofer IESE (Germany)
Günther Rue, University of Calgary (Canada)
Ioana Rus, Fraunhofer, Maryland (USA)
Kurt Schneider, DaimlerChrysler (Germany)
Carsten Tautz, empolis (Germany)
Marvin Zelkowitz, University of Maryland (USA)

Table of Contents

Learning Software Organizations and Agile Software Development: Complementary or Contradictory Concepts?

Scott Henninger[1] and Frank Maurer[2]

[1] Department of Computer Science & Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA
scotth@cse.unl.edu
[2] Department of Computer Science, University of Calgary, Calgary, Alberta T2N 1N4, Canada
maurer@cspc.ucalgary.ca

The LSO Workshop series has focused on technical, organizational, and social solutions to problems of learning from past experiences and codifying the resulting best practices so they can be systematically used in subsequent software development efforts. The theme of the Fourth Workshop on Learning Software Organizations (LSO), held in Chicago, Illinois in August of 2002 was "Balancing Agile Processes and Long-Term Learning in Software Organizations". In this workshop, we wanted to explore the issues involved in managing knowledge in dynamic domains requiring significant levels of improvisational change within each repetition of the process [1]. Thus, it seemed appropriate to collocate with the XP Agile Universe conference, which focuses on Agile development methods that require significant levels of flexibility in the development process.

In this light, the challenge for the LSO community becomes one of understanding how the lessons learned and knowledge constructed in past projects can be brought to bear on subsequent software development efforts. The LSO community must therefore balance the capture and dissemination of knowledge with necessary flexibility that enhances the ability of an organization to quickly adapt its processes to new technologies and market pressures [2].

The Agile community, on the other hand, is faced with the opposite problem. Contrary to popular belief, the Agile community is not against documentation, but is prudently suspicious of over-documentation, especially when requirements are volatile [3]. But this may lead to situations where institutional knowledge is lost when people move on, are moved to new projects, or the size of an organization becomes big enough to impede the low of knowledge, making it difficult to improve or avoid repeating failures. The agile approach of knowledge sharing by face-to-face communication also limits learning to occur between independent teams in a development organization.

So the problems facing these communities seem complementary. At its core, the desire is to design work practices that balance the desire for innovation with knowledge of past experiences and best practices that will enable improvement in quality, productivity, and etc. This tension between past knowledge and innovation is particularly acute in the software industry, which involves the development of a highly variable product that dictates the need for continuous process adjustments.

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 1–3, 2003.

The Agile community has focused on innovation and transfer of tacit knowledge, while the LSO community has focused on documenting and externalizing knowledge, or best practices. Each community has a piece of the puzzle, but neither have a full solution.

The workshop, which led to the papers in this book, explored issues ranging from how to use tool support to integrate organizational learning and agile processes to experience reports on using LSO and knowledge management repositories.

The paper "The Collaborative Learning Methodology CORONET-Train: Implementation and Guidance" by Niniek Angkasaputra, Dietmar Pfahl, Eric Ras, and Sonja Trapp reports on how to integrate web-based learning into continuous learning processes at a software organization. It describes a framework for learning and reports on an industrial case study that applied the approach.

In their paper, "Extending Agile Methods: Postmortem Reviews as Extended Feedback," Torgeir Dingsøyr and Geir Kjetil Hanssen describe how lightweight postmortem project reviews can be used to improve agile team processes. The paper follows similar lines of thinking as some work by Alistair Cockburn from the agile community.

Raimund L. Feldmann and Markus Pizka describe a distributed software engineering portal in their paper "An On-line Software Engineering Repository for Germany's SME – An Experience Report". The portal targets to support small and medium size software organizations that often do not have the resources to maintain a SE repository for themselves.

In "Tool Support for Experience-Based Methodologies," Scott Henninger demonstrates a tool-based approach to organizational learning through the BORE (Building an Organizational Repository of Experiences). Bore combines a flexible software process approach to tailor knowledge to the needs of a project with an experience-based feedback mechanism to promote organizational learning. Rule-based and case-based reasoning approaches are used to collect and disseminate software knowledge.

Learning in software development is often based on communities of practice. IN "Building Communities among Software Engineers: The ViSEK Approach to Intra- and Inter-Organizational Learning," Britta Hofmann and Volker Wulf explain how the formation of these communities can be enhanced by combining a software engineering portal with the creation of regional networks of software developers.

The paper by Harald Holz and Frank Maurer on "Knowledge Management Support for Distributed Agile Software Processes" present a lightweight approach for process-centered experience management. It reduces the knowledge maintenance problem by utilizing existing web-based resources and only modeling the information needs of team members.

Mikael Lindvall, Ioana Rus, and Sachin Suman Sinha in "Technology Support for Knowledge Management" provide an overview on tools and approaches for supporting learning in an organization. They discuss a broad range of technologies covering document management systems, collaboration services, expert finder approaches and others.

The paper "Learning and Understanding a Software Process through Simulation of its Underlying Model" by Holger Neu and Ulrike Becker-Kornstaedt present an approach that uses discrete event simulation of a software process to help students to learn about the effects of changing decisions.

The last paper “Software Engineering Decision Making Support – A New Paradigm for Software Organizations” by Günther Ruhe shines a new perspective on software development. It addresses software processes from the decision support perspective and shows that a successful project relies on providing the right knowledge to the right people at the right time.

Overall, the papers in this volume reflect the current state of the art and major future trends in learning software organizations. These techniques have potential to have a significant impact on software development practices in the future, improving both software development productivity and quality by learning from past experiences.

References

1. T. Dybå, "Improvisation in Small Software Organizations," *IEEE Software*, vol. 17, pp. 82-87, 2000.
2. J. A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York: Dorset House, 2000.
3. A. Cockburn and J. A. Highsmith, "Agile Software Development: The People Factor," *IEEE Computer*, vol. 34, pp. 131-133, 2001.

Extending Agile Methods: Postmortem Reviews as Extended Feedback

Torgeir Dingsøyr and Geir Kjetil Hanssen

SINTEF Telecom and Informatics,
7465 Trondheim, Norway
{Torgeir.Dingsoyr,Geir.K.Hanssen}@informatics.sintef.no

Abstract. Agile software development methods, such as Extreme Programming, focus on informal learning mechanisms like pair programming. Yet powerful methods, new knowledge that is gained in a project will not spread rapidly in an organisation if knowledge and experience is not externalised. We propose to combine a lightweight externalisation method: postmortem reviews with agile methods to strengthen the overall learning, and suggest how this can be done. We use practical experience from an Extreme Programming development project, and from conducting postmortem analysis in several companies in our discussion.

Keywords: Agile Development Methods, Extreme Programming, Postmortem reviews, Knowledge Management, Experience Elicitation.

1 Introduction

Agile software development methods such as Extreme Programming (XP) have in the past years achieved an explosive interest in the international information technology community. This can be seen as a reaction to the more traditional and control-oriented methods, where the waterfall method is an example. These methods focus on predictability through extensive planning and thoroughly follow ups on the plans. This way of running projects and develop software conflicts with the increasing need to handle rapid changes in projects and their environment. It is here that agile methods fit in. They are built to handle changes in design and requirements [1]. They open up for creativity during the whole project lifecycle and not just in the initial planning and design phases. Agile means smooth, non-bureaucratic and adaptable.

There are several methods in the agile family, for example Scrum, Crystal, FDD and more. Looking into these methods support for project and organization learning would be a comprehensive task. We have chosen to look at Extreme Programming, because this is a popular method.

The learning processes in XP are, as the rest of the method, agile. That means that the learning process is simplified compared to other more comprehensive development methodologies.

Pair programming is one of the XP "practices" that are designed to support knowledge transfer within a team. This works well for pairs of programmers, but we claim that learning can be even better if this is combined with other learning modes. In this paper we suggest to extend XP with an "agile" method for harvesting knowledge from

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 4–12, 2003.

a group and documenting them in a way that makes them accessible to others: using postmortem reviews. We will show an example of a student XP project where postmortems have been used.

Now, we first discuss learning in software development, and outline a framework of learning modes developed by Nonaka and Takeuchi. Then, we discuss how learning takes place now in XP, before briefly discussing the research method applied here. Further, we introduce postmortem analysis, and show from a student example how this can be applied in XP. We end by discussing the additional learning aspects introduced by the postmortem.

2 Learning in Software Development

There are many theories of how learning takes place in ordinary work situations. Brown and Dugid [2] have described how "communities of practise" learn from each other, David Kolb [3] describes how people learns from experience, and Nonaka and Takeuchi [4] have developed a "theory of knowledge construction".

Of course, learning also takes place in software development. New technologies, work methods as well as problems in existing software development methods requires that software developers are keen on learning.

2.1 A Model of Learning

To discuss learning in software engineering, we will rely on the model presented by Nonaka and Takeuchi. We choose this model, because of its emphasis on tacit knowledge, which shows how learning takes place in agile development methods. It is also a model that is generally known. They divide between two types of knowledge: *Tacit* knowledge, that humans have, but are unable to represent. An example is "how to ride a bike" – which is difficult if not impossible to write down. The other type of knowledge is *explicit* knowledge – which is possible to represent as process guidelines or any other form of documentation.

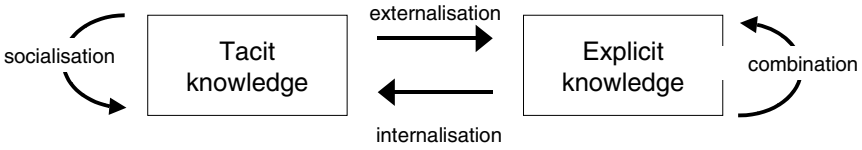

Fig. 1. Conversion of Tacit and Explicit Knowledge.

Further, Nonaka and Takeuchi divide between four modes of knowledge conversion between tacit and explicit knowledge, as shown in Figure 1:

- *Socialisation* means to transfer tacit knowledge to tacit through observation, imitation and practice, what has been referred to as "on the job" training. Craftsmanship has usually been learned in this way, where oral communication is either not used or plays a minor part.

- *Internalisation* is to take externalised knowledge and make it into individual tacit knowledge in the form of mental models or technical know-how. “Documents and manuals facilitate the transfer of explicit knowledge to other people, thereby helping them experience the experiences of others indirectly (i.e. 're-experience' them)”.
- *Externalisation* means to go from tacit knowledge to explicit. Explicit knowledge can “take the shapes of metaphors, analogies, concepts, hypotheses, or models”. This conversion is usually triggered by dialogue or collective reflection, but can also be the result of individual reflection, for example in a writing process.
- *Combination* is to go from explicit to explicit knowledge, that is, to combine and systematise knowledge from different sources such as documents, meetings, telephone conferences, or bulletin boards. Systematising this kind of explicit knowledge is to reconfigure it by sorting, adding, combining, or categorising the knowledge.

According to Nonaka and Takeuchi knowledge passes through different modes of conversion in which makes the knowledge more refined, and also spreads it across different layers in an organisation.

We will be using this framework in our discussion to come.

2.2 Learning in Extreme Programming

XP emphasises on teamwork and communication within software development teams. The originator of XP, Kent Beck, claims that the bottlenecks in software development all stem from the difficulty in communication between people [5]. One of the 13 practices in XP [1], pair programming, is designed to ensure that the programmers learn from each other. Pair programming is a technique that supports transfer of tacit knowledge, as Barry Boehm point out in an article, “agile methods derive much of their agility by relying on the tacit knowledge embodied in the team, rather than writing the knowledge down in plans” [6]. This can be general programming knowledge, knowledge on the use of tools, knowledge of the application domain etc. Pair programming works as knowledge transfer as what is called socialisation in Nonakas model.

XP in its simplest form prescribes no other documentation than the source code, which in most cases means that much relevant knowledge is inaccessible to others than the ones that work in each project.. Nevertheless, XP projects often develop other types of documentation, but this is often on a 'need to know' basis and is not intended to be used for knowledge transfer beyond the project.

3 Research Methods

The research reported here was carried out in a research project with participants from academia and industry, called Process Improvement for IT industry (PROFIT). Researchers work in tight co-operation with nine companies that develop software, with different process improvement initiatives.

For the part on postmortems, we have participated in collecting experience from real software projects in a real environment. This is often referred to as action research [7, 8]. The benefit with this type of research is that the actual problems are

very relevant to industry. A difficulty is that we have limited control over the surroundings, so the results might not be as generally applicable as when using for example experiments.

The researchers have had two roles in this work: First, as a kind of process leader who have organised a meeting: Set the agenda in co-operation with industry and organised discussions. On the other hand, the researchers have been observers trying to document the process and the results of the meeting.

For the part on XP, we have run a student project using XP, where four students have been working in couples for 12 weeks to develop software. We have had weekly meetings with the students to discuss progress. Our analysis and description in this paper are based on discussions with these students in addition to the literature.

4 Learning through Postmortem Reviews

A postmortem review is a review done in the end of a project to sum up good and bad experiences. There are several ways to perform such [9]. Apple has used a method [10] which includes designing a project survey, collecting objective project information, conducting a debriefing meeting, a "project history day" and finally publishing the results. At Microsoft they put much effort into writing "postmortem reports". These contain discussion on "what worked well in the last project, what did not work well, and what the group should do to improve in the next project" [11]. The size of the resulting documents are quite large, "groups generally take three to six months to put a postmortem document together. The documents have ranged from under 10 to more than 100 pages, and have tended to grow in length".

A problem with these approaches is that they are made for very large companies, who can spend a lot of resources on analysing completed projects. We work with medium-sized companies where 5-10 people usually participate in a project, ranging in size from about 8 to 50 man-months. To suit this type of projects, we have developed a "lightweight" version of postmortem reviews [12, 13].

We have used postmortem reviews as a group process, where most of the work is done in one meeting lasting only half a day. We get as many as possible of the people who have been working in the project to participate, together with two researchers, one in charge of the postmortem process, the other acting as a secretary. The goal of this meeting is to collect information from the participants, make them discuss the way the project was carried out, and also to analyse causes for why things worked out well or did not work out.

Our "requirements" for this process is that it should not take much time for the project team to participate, and it should document the most important experience from the project, together with an analysis of this experience.

4.1 Lightweight Postmortem Review

We have used two techniques to carry out lightweight postmortem reviews. For a focused brainstorm on what happened in the project, we used a technique named after a Japanese ethnologist, Jiro Kawakita [14] – called "the KJ Method". For each of these sessions, we give the participants a set of post-it notes, and ask them to write one "issue" on each. We usually hand out between three and five notes per person,

depending on the number of participants. After some minutes, we ask one of them to attach one note to a whiteboard and say why this issue was important. Then the next person would present a note and so on until all the notes are on the whiteboard. The notes are then grouped, and each group is given a new name.

We use a technique for Root Cause Analysis, called Ishikawa or fishbone-diagrams to analyse the causes of important issues. We draw an arrow on a whiteboard indicating the issue being discussed, and attach other arrows to this one like in a fishbone with issues the participants think cause the first issue. Sometimes, we also think about what was the underlying reasons for some of the main causes and attach those as well.

4.2 Example Results from a Lightweight Postmortem Review

The following example of KJ and Root Cause Analysis is taken from a non-XP project, but the example illustrates the techniques, which is independent of the method. One result from a KJ session was two post-it notes grouped together and named "changing requirements", with the following statements from two developers:

"Another thing was changes of requirements during the project: from my point of view – who implemented things, it was difficult to decide: when have the requirements changed so much that things have to be made from scratch? Some wrong decisions were taken that reduced the quality of the software".

"Unclear customer requirements – which made us use a lot of time in discussions and meetings with the customer to get things right, which made us spend a lot of time because the customer did not do good enough work."

When we later brought this up again and tried to find some of the root causes for "changing requirements", we ended up with the fishbone diagram in Fig. 2.

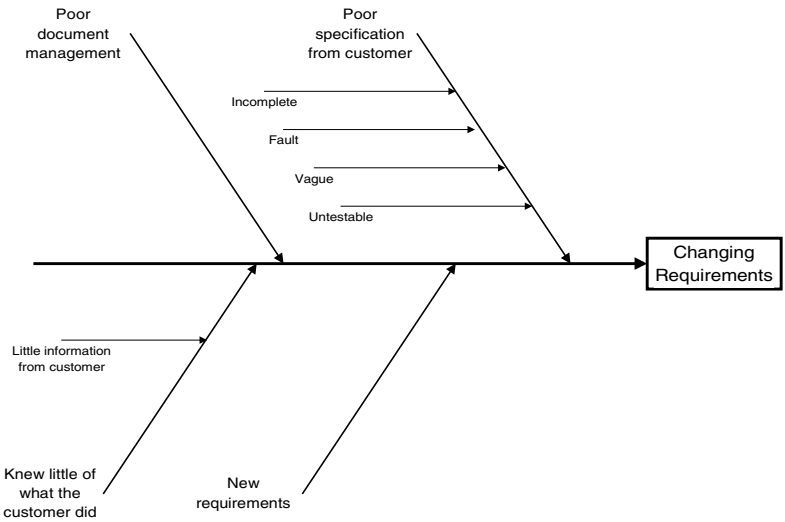

Fig. 2. Fishbone diagram for "Changing Requirements".

The root causes for the changing requirements, as the people participating in the analysis saw it, was that the requirements were poorly specified by the customer, there were “new requirements” during the project, and the company knew little of what the customer was doing. Another reason for this problem was that documents related to requirements were managed poorly within the company. In Fig. 2, we have also listed some subcauses.

4.3 Experience with Postmortem Reviews in Extreme Programming

Two students used postmortem reviews as a learning mechanism in XP. They developed an Intranet-based electronic process guide for a local company, and used XP as their development process. The project lasted for three months and the two programmers used approximately 430 hours on the development project. The project had 7 iterations, and all XP practices were followed. In addition to the project work they also spent approximately 20 hours on postmortem activities to collect and discuss experiences.

Initially the programmers used XP as it is described in the literature [15]. Although XP is built on many well known concepts, it needs local adaptation and tailoring [5]. With this in mind the students wanted to adapt their use of XP during the project, based on their own experiences. The practises were revised from iteration to iteration. After each of the seven iterations a postmortem was conducted to identify both positive and negative experiences. The participants were the two programmers and the customer representative; the whole session lasted for about one hour. Examples of positive experiences collected from such postmortems were *"spiking on new technology is useful"* and *"refactoring is useful in each iteration"*. Examples of negative experiences were *"get no production code when one programmer is sick (inactive)"* and *"difficult to estimate large stories"*. The programmers used these experiences to suggest improvements in the XP-practices. An example of one such improvement suggestion is *"must split large stories into small ones, maximum 1/2 an iteration"*.

Initially each practice was printed on a paper and taped to a wall; each sheet had a description of one practice. The most interesting post-it notes (the experiences) from the postmortem were placed beside the sheet describing the related practice. In the same way, each improvement suggestion was written on a post-it note and placed beside the related practice. The programmers also wrote suggestions during the iteration. When the programmers decided to implement a suggestion, they removed the suggestion post-it note and adjusted the practice by rewriting the practice description, print it, and then place it on top of the old sheet. In this way they had a continuously experience based adaptation of XP. At the end of the project they had all changes placed on the wall and they could look back on the improvement history of each practice by looking at the old sheets below the prevailing one on the top. In Figure 3, we show how the "improvement wall" looked like during the project.

Figure 4 shows the number of suggestions that came up from the postmortem on each iteration. Most of these suggestions were implemented in the next iteration.

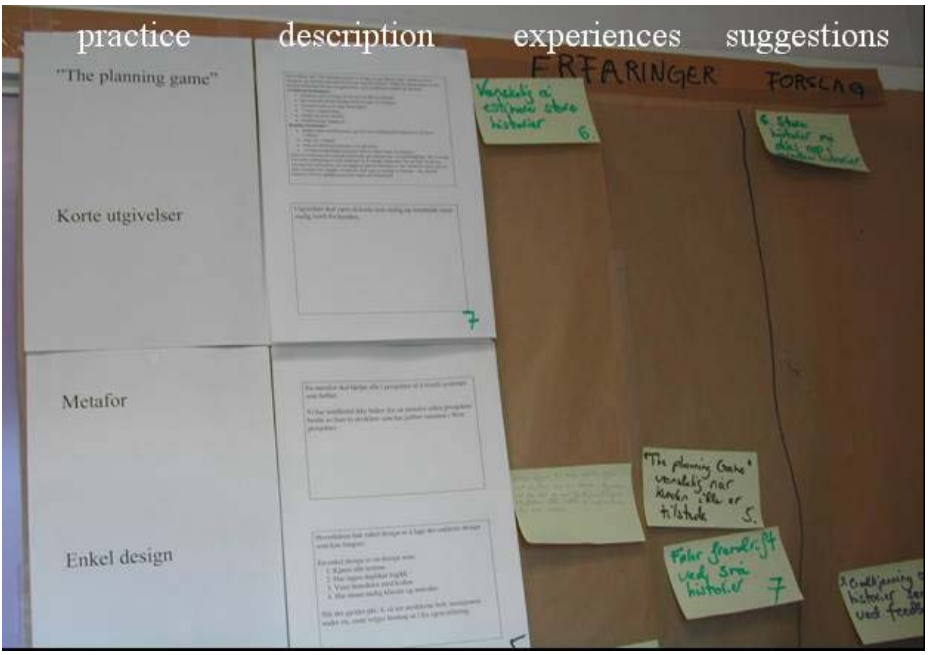

Fig. 3. The improvement wall, showing practice names and descriptions, experience and suggestions for changes (picture text in Norwegian).

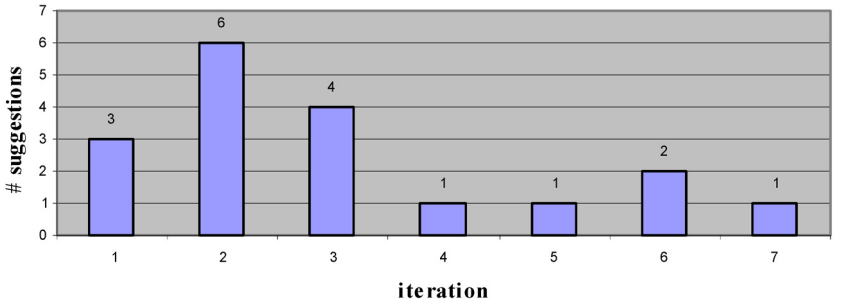

Fig. 4. Improvement suggestions per iteration.

5 Discussion and Conclusion

This paper argues that postmortem reviews and especially the KJ-technique can be used to adapt XP during a project. It is likely to believe that some of these experiences and the suggestions from the student project never would have been found without this support. One advantage of using KJ in this case was that the programmers could evaluate their experiences in co-operation with other stakeholders in the project, which in this case was the customer representative. Other stakeholders like system

architects, document and database-responsible can also be involved to a greater extent.

We have now shown postmortem reviews as a method for capturing experience from software projects, and have shown results from applying it in a student XP project.

Pair programming seems to be an excellent method for socialisation in software development projects, but we think that learning gets even better if this method is combined with others, as shown in Figure 5. We have shown how XP can be extended in a quite "agile" way to also support externalisation.

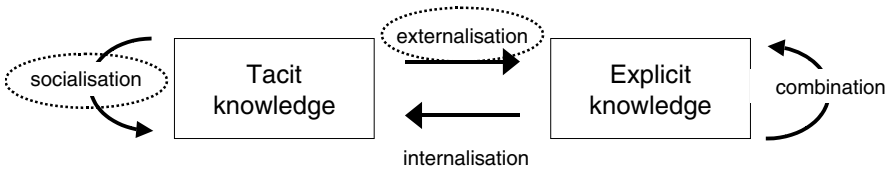

Fig. 5. Extending XP for learning with both socialisation and externalisation.

This extension increases learning in the following ways:

- Other people than the ones participating in an XP project can gain knowledge on XP, and especially how XP is best tailored to an organisation's needs.
- The people participating in an XP project will gain a deeper knowledge of their own XP practise by externalising knowledge – and can more easily discuss changes in practise.

  We think this is a method that can give new insights on local adaptations of software development, give better educated software engineers, and make XP more varied and fun for software developers.

Acknowledgements

We would like to thank Nils Brede Moe from SINTEF Telecom and Informatics and Tor Stålhane from the Norwegian University of Science and Technology for setting up student experiments on XP. We would further like to thank the students Håvard Julsrud Hauge, Øivind Mollan, Ole Johan Lefstad, Sverre Stornes and Jørgen Brudal Sandnes, all from the Norwegian University of Science and Technology, for discussions and documented experience on XP practise.

References

1. K. Beck, "Embracing Change with Extreme Programming," *IEEE Computer*, pp. 70 - 77, October, 1999.
2. J. S. Brown and P. Duguid, *The Social Life of Information*. Boston, Massachusets: Harvard Business School Press, 2000.

3. D. Kolb, *Experiental Learning: Experience as the Source of Learning and Development*: Prentice Hall, 1984.
4. I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company*: Oxford University Press, 1995.
5. K. Beck, "Extreme Programming: A Humanistic Discipline in Software Development," presented at Fundamental Approaches to Software Engineering. First International Conference (FASE'98), 1998.
6. B. Boehm, "Get Ready for Agile Methods, with Care," *IEEE Computer*, pp. 64 - 69, January, 2002.
7. D. J. Greenwood and M. Levin, *Introduction to Action Research*: Sage Publications, 1998.
8. D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM*, vol. 42, pp. 94-97, 1, 1999.
9. A. Birk, T. Dingsøyr, and T. Stålhane, "Postmortem: Never leave a project without it," *IEEE Software, special issue on knowledge management in software engineering*, pp. 43-45, May/June, 2002.
10. B. Collier, T. DeMarco, and P. Fearey, "A Defined Process For Project Post Mortem Review," *IEEE Software*, pp. 65-72, July, 1996.
11. M. A. Cusomano and R. W. Selby, *Microsoft Secrets - How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*: The Free Press, 1995.
12. T. Stålhane, T. Dingsøyr, N. B. Moe, and G. K. Hanssen, "Post Mortem - An Assessment of Two Approaches," presented at EuroSPI, Limrerick, Ireland, 2001.
13. T. Dingsøyr, N. B. Moe, and Ø. Nytrø, "Augmenting Experience Reports with Lightweight Postmortem Reviews," presented at Third International Conference on Product Focused Software Process Improvement, Kaiserslautern, Germany, 2001.
14. R. Scupin, "The KJ Method: A Technique for Analyzing Data Derived from Japanese ethnology," *Human Organization*, vol. 56, pp. 233-237, 2, 1997.
15. K. Beck, *Extreme Programming Explained*: Addison-Wesley, 2000.

The Collaborative Learning Methodology CORONET-Train: Implementation and Guidance

Niniek Angkasaputra, Dietmar Pfahl, Eric Ras, and Sonja Trapp

Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
{angkasa,pfahl,ras,trapp}@iese.fraunhofer.de

Abstract. The introduction of new methodologies into a software organization is a non-trivial problem because it usually requires the alteration of existing business and development processes and the change of habits and beliefs of software engineers and managers. This is particularly true for comprehensive methodologies that integrate e-learning and knowledge management. An example is the web-based collaborative learning methodology CORONET-Train recently developed in an international research project. This paper describes how CORONET-Train and its supporting software infrastructure WBT-Master can be successfully established and retained in a software organization by using adequate implementation and guidance processes, which take into account aspects of organizational culture. In addition, the paper summarizes results gained from industrial case studies that provide evidence for the effectiveness of these processes.

Keywords: Collaborative Learning, CORONET, Organizational Culture, Knowledge Management, WBT-Master

1 Introduction

Continuous education has become an important facilitator of software process improvement [1] and organizational learning [2, 8]. However, university education and classroom-based professional training courses alone cannot satisfy the constantly growing demand for specific technology training in software industry due to the limited number of people they can reach. New methodologies and software infrastructures for learning at the workplace are needed.

The European research project CORONET (Corporate Software Engineering Knowledge Networks for Improved Training of the Workforce), which finished in April 2002, provides the comprehensive learning methodology CORONET-Train. The overall goal of CORONET-Train is to facilitate web-based collaborative learning of software engineers at the workplace in a systematic and on-demand-focused way.

The introduction of complex new methodologies such as CORONET-Train into a software organization is a non-trivial problem because it affects the existing business and development processes, and it requires the change of established habits and beliefs of software engineers and managers. In particular, acceptance of the software infrastructure needed to support the new learning methodology and compatibility of the methodology with the organizational culture, i.e. with regard to collaboration, require special attention to avoid that they impede successful introduction.

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 13–24, 2003.

Once introduced into the organization, another risk for the sustained implementation of a learning methodology such as CORONET-Train is the lack of guidance for the users of the methodology, i.e. the software engineers, and those who are in charge of managing the whole learning environment, i.e. the learning managers. Therefore, CORONET-Train offers a framework and a guide map that help to match available learning scenarios with emerging learning needs.

This paper describes how CORONET-Train and its supporting software infrastructure can be successfully established and retained in a software organization by using adequate implementation and guidance processes, and taking into account the maturity of the collaborative culture within the organization.

The structure of this paper is as follows. In Section 2, a brief description of CORONET-Train is given, and its recommended supporting infrastructure WBT-Master is presented. In Section 3, the importance of organizational culture for the establishment of collaboration is discussed. The processes for implementing and operating CORONET-Train, which are the focus of this paper, are described in Section 4. Results of industrial applications and evaluation studies are presented in Section 5. Conclusions are presented in Section 6.

2 CORONET-Train

This section briefly outlines the learning methodology CORONET-Train, highlighting the collaborative aspects, the set of learning, knowledge transfer and knowledge engineering methods, the integration of e-learning with knowledge management, and the recommended software infrastructure, WBT-Master. More complete descriptions of CORONET-Train and WBT-Master can be found in [6, 11].

2.1 Collaborative Learning with CORONET-Train

In industrial software organizations, there is a broad range of learning settings[1] that potentially apply to knowledge workers who are in need of evolving their professional knowledge and skills. One extreme of the range is the learning setting ‘participating in a course’, characterized by the organizational need for long-term competence development with predefined individual learning goals, well-structured subject matters, and availability of dedicated trainers/tutors and learning materials. On the other extreme, there is the setting ‘learning within daily work’, characterized by short-lived learning goals of knowledge engineers and the need for spontaneous information search, mainly aimed at solving problems that emerge from daily work.

A major strength of CORONET-Train is its ability to cover the whole bandwidth of learning settings, from short-term problem-solving through quick information access to long-term competence development through dedicated web-based training, tutoring and mentoring. The important characteristic of CORONET-Train is its focus on collaborative approaches for all relevant learning settings. In particular, CORONET-Train promotes and supports the development of sustained interpersonal

[1] The term ‘learning setting’ in this paper is defined as the implementation of a didactical design consisting of topic-related content, instructional strategies, learning activities, and tool support.

relationships in combination with comprehensive functionality for accessing, annotating, and extending materials (from others and for others). In this way, it helps to establish learning networks in which people of equal and different competence levels practice both individual and group learning, experience-based learning, learning with multiple activities and resources, and knowledge sharing.

2.2 CORONET-Train Methods and Learning Scenarios

In order to facilitate collaborative learning, CORONET-Train offers three classes of methods, each method consisting of a set of processes and activities (see Fig. 1):

a) *Learning methods*: Five methods (Case-Based Learning, Theme-Based Learning, Web-Based Training, Web-Based Tutoring, and Knowledge Sharing) define learning processes[2] and activities that are adequately tailored to specific learning situations and learning needs of software engineers. The description of processes and activities is made from the perspective of those who wish to acquire new knowledge and skills related to a specific subject matter.
b) *Knowledge transfer methods*: Three methods (Training, Tutoring, Mentoring) define processes and activities that subject matter experts can apply in order to disseminate their know-how and help software engineers satisfy their learning needs.
c) *Knowledge engineering methods*: Four methods (Authoring, Structuring, Administration, Management) define processes and activities that are needed to develop, structure, and maintain learning resources, to set-up and maintain the software infrastructure, to administer the users of the infrastructure, and to introduce and manage the learning environment.

A learning scenario is an implementation of one or more CORONET-Train methods or parts of them (i.e. processes and their activities). In a learning scenario, processes and activities are adapted to a particular learning situation and supporting software infrastructure. The purpose of learning scenarios is to organize and maintain relationships among individuals involved in a learning situation by defining the sequence of tasks and their associated actions, which have to be performed in order to reach a learning objective.

During the CORONET project, a number of learning scenarios have been developed in order to show how CORONET-Train can be applied to frequently occurring learning situations in software organizations, e.g. web-based mentoring, web-based knowledge mining, web-based collaborative problem-solving, and web-based virtual classroom.

Once learning managers are familiar with CORONET-Train, they can develop new learning scenarios to adapt CORONET-Train to different learning situations. The following steps can help to develop a particular learning scenario:

a) describe the current learning situation;
b) characterize the type of learning (e.g. learning a subject matter topic comprehensively, problem solving, etc.);
c) identify relevant means to support the learning (e.g. learning platform, internet technology, knowledge resources etc.);

---

[2] In CORONET-Train, the term ‘learning process’ is used to define a sequence of learning activities. This differs from the usage of the term ‘learning process’ in educational science, where it refers to the internal processing of information by a learner.

d) select appropriate processes and activities from the CORONET-Train methods;
e) represent the learning scenario in the form of a guidebook.

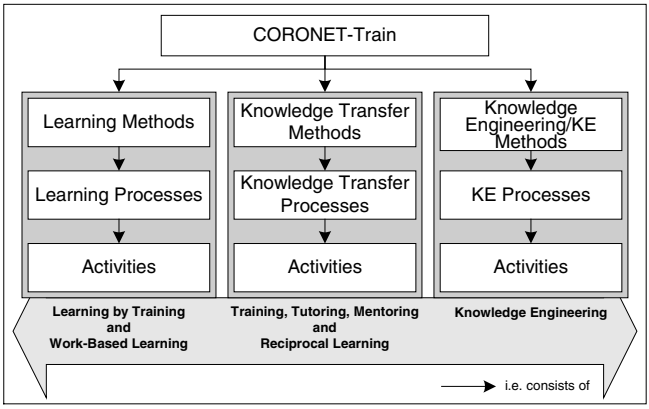

Fig. 1. Overview of CORONET-Train

2.3 Integration of e-Learning (Web-Based) and Knowledge Management

The integration of e-learning and knowledge management processes has the potential to produce synergies that significantly improve the creation of new (tacit and explicit) knowledge, as well as the performance of efficient and effective learning processes.

Core processes of knowledge management are knowledge creation, knowledge structuring, and knowledge dissemination (see Fig. 2). Knowledge dissemination and knowledge creation also occur in e-learning. Typically, however, in e-learning, the links between knowledge dissemination and knowledge creation are rather weak from an organizational perspective. On the one hand, new (individual) knowledge generated through learning is not (sufficiently) made explicit, e.g. in the form of new or enhanced learning materials. On the other hand, there is a lack of adequate structuring mechanisms that would allow for easy retrieval and reuse of existing knowledge (generated by others).

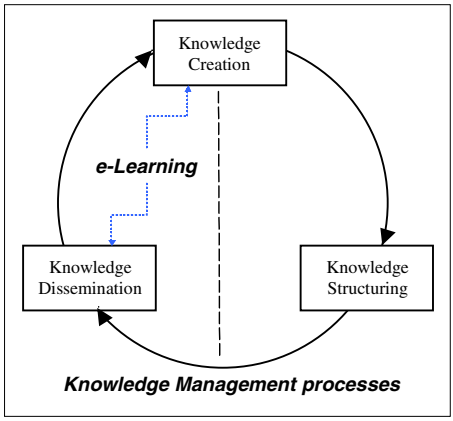

Fig. 2. Integration of e-learning with knowledge management in CORONET-Train

CORONET-Train bridges this gap by providing flexible annotation and recording mechanisms for discussion threads, and by applying advanced knowledge structuring methods for all sorts of learning resources. Application of these mechanisms facilitates on-demand creation, evaluation and evolution of learning materials based on effective retrieval, feedback, and reuse support. In addition, knowledge creation, knowledge structuring and knowledge dissemination can benefit from learning networks and their associated collaborative learning processes by systematically capitalizing upon individual expertise gained during collaborative learning.

2.4 Supporting Infrastructure: WBT-Master

In order to put CORONET-Train into practice and to be able to evaluate its concepts within the CORONET project, the prototype of a supporting infrastructure was developed under the name WBT-Master.

WBT-Master provides essential features to support communication and collaboration, e.g., comprehensive access mechanisms to relevant knowledge repositories, subject matter experts and peer learners.

Technically speaking, WBT-Master is a complex Internet application that supports several dozens of different tools, utilizes many different data structuring facilities, and can (potentially) provide access to limitless different learning resources. At the same time, the system is highly modular and just a few components might be dynamically selected to work with (personal desktop concept). On the other hand, WBT-Master integrates all its tools and processes into a single operating environment. Moreover, any learning object produced by means of one system component by one user may be reused by others in different contexts. Thus, even such non-innovative components as structured discussion forums, and annotations get new value just because of the possibility to seamless integration with other components.

Another strength of WBT-Master is its high degree of compatibility. A major problem of modern e-learning solutions is content incompatibility (of proprietary content). Users are often reluctant to switch to new, more advanced e-learning platforms because of the presence of existing content, which might be incompatible with the new system. WBT-Master supports internationally recognized standards as IMS/LRN 2.0 and SCORM, and special converting tools to import/export any content elements compatible with the standards.

3 Organizational Culture and Collaborative Learning

Based on the definition by E.H. Schein [9], culture is recognized as an invisible medium that forms our communications. It determines our definitions of success, prosperity, growth, failure, mediocrity, and effectiveness. The cultural aspects that help to define an organizational culture are artifacts, espoused beliefs and basic underlying assumptions of the organization.

Collaboration is a natural social skill that enables people to accomplish more than they would accomplish alone. Collaboration is a product of one or more common goals, values, needs, ideals, visions or interests.

Organizational culture has some bearing on the establishment of collaboration. Collaborative culture is characterized by constructive interdependence, heterogeneity,

and personal competencies for learning, teaching/training and problem-solving. There are three defined maturity levels of collaborative culture:

I. An organization in which individuals collaborate on an ad-hoc, reactive basis, i.e., no knowledge processes or systems for tracking of knowledge exist.
II. An organization that has some success in collaboration, training programs on collaboration and 'best practices' learning.
III. An organization that has a collaborative function, relationship database, proactive collaboration strategy and knowledge programs tied to rewards.

The CORONET system, formed by the methodology CORONET-Train and its supportive infrastructure WBT-Master, provides comprehensive solutions to establish collaborative learning in software organizations. But its effectiveness partly depends on the existence of a collaborative culture within the software organization. Since high-level collaborative culture makes it easy to establish learning networks, learning managers should concentrate on initiating the cultural changes that are necessary for the introduction and maintenance of collaborative learning processes. On the other hand, the implementation of CORONET-Train and WBT-Master is not restricted to maturity level III organizations. Due to its modularity, the CORONET system can easily be adapted to all levels of collaborative culture by the possibility to implement only subsets of its methods, processes, and tools. Nevertheless, an organization should strive towards achieving level III of collaborative culture, in order to fully benefit from all the functionality designed especially for collaborative learning.

4 Implementing CORONET-Train

The implementation of CORONET-Train consists of two main stages, the introduction stage and the operational stage. The introduction stage is necessary for a software organization which uses CORONET-Train for the first time, while the operational stage is important to further establish and expand the usage of CORONET-Train within an organization after introduction.

4.1 Introduction Stage

In order to introduce CORONET-Train into a software organization four steps need to be taken:

Step 1 (Preparation): The strategic manager of the organization identifies the objectives and scope of learning and assigns persons to the CORONET roles Learning Manager, Author, Knowledge Engineer, and Administrator[3].

Step 2 (Set-up): The learning manager identifies potential users, defines and assigns suitable competence levels, identifies and selects existing (pre-defined) learning scenarios or develops new learning scenarios. The Administrator installs the learning infrastructure (e.g. WBT-Master), administrates the system (i.e. configures the repository, user access, etc.). Knowledge Engineer and Author export existing corporate learning resources to the repository and develop training material.

[3] The CORONET role model is fully explained in [6].

Step 3 (Execution): The learning manager decides on the proprietary learning network (consisting of learning groups and relevant contents) and guides the Knowledge Workers in performing the learning scenario.

Step 4 (Dissemination): The strategic and learning managers initiate the rollout of CORONET-Train within the organization.

During the introduction stage, special care should be given to the users (i.e. the Knowledge Workers). The following success factors should be considered:

a) Start with simple learning scenarios, and then gradually develop more complex learning scenarios.

b) Provide support based on the three types of new users: (a) users who are novices of using the methodology and infrastructure and are experts of subject matters, (b) users who are familiar with the methodology and infrastructure but novices of subject matters (c) users who are novices of both subject matters and the methodology and infrastructure.

c) Provide adequate guidance until the learning objective is achieved and users are getting accustomed to the new learning environment.

d) Give the users the chance to understand how the system works:
   - Let the users experience all of the learning and knowledge transfer methods
   - Give them time to grasp the typical features of the infrastructure functionality
   - Afterwards, rely on the intelligence and the intuition of the users to creatively use the system.

e) In order to attract attention, provide adequate contents that respond to the competence levels of the users.

It should be underlined that the gradual introduction is the best way for software engineers to adapt to the learning methodology and infrastructure, starting from simple learning scenarios that are customized to the typical learning needs. Having experienced at least one simple learning scenario, users are expected to be ready to proceed to other more complex learning scenarios.

4.2 Operational Stage

Once the introduction stage is over, regular operation of the CORONET system starts. In the operational stage, the Knowledge Workers are acquainted to the new way of learning and can continue without close involvement of the Learning Manager. The Learning Manager interferes only when necessary.

To make sure that collaborative learning according to CORONET-Train further develops throughout the organization, sufficient guidance of the system users is needed. Thus, during the operational stage, a learning cycle consisting of three phases guides the Knowledge Workers as shown in Fig. 3.

Phase 1: Identification of learning need
- Specification of problems to be solved, knowledge to be acquired, or ideas to be discussed
- Retrieval of available information and identification of peers/experts

Phase 2: Collaborative learning-on-demand
- Choice of suitable learning scenarios
- Execution of learning scenarios
- Generation of solution, acquirement of knowledge

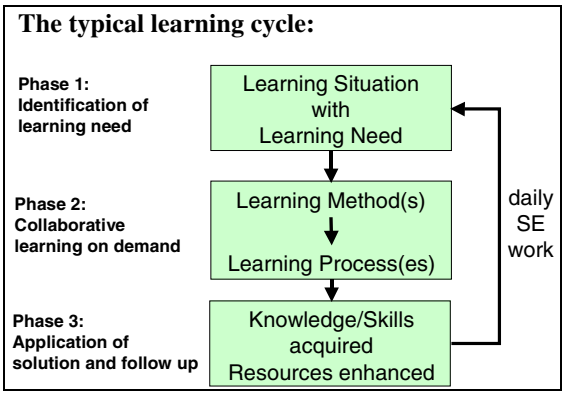

Fig. 3. The three-phase CORONET-Train learning cycle during the operational stage

Phase 3: Application of solution and follow up

- Documentation of results (knowledge or problem solution)
- Application of the acquired knowledge or problem solution
- Validation of knowledge or problem solution and feedback

4.3 CORONET-Train Application Guide Map

To support the choice of a suitable learning scenario, the definition of a guide map is strongly encouraged. Using the symbols shown in Fig. 4, CORONET-Train provides a complete guide map[4] comprising all pre-defined learning scenarios. With the help of this guide map, it is easy to find the adequate scenario for a particular learning need in a specific learning situation.

action	Contains a phrase to indicate an action performed by a Knowledge Worker in phase 1 of learning cycle.
state	Describes a state of a Knowledge Worker's situation.
learning scenario	Indicates which learning scenario is being applied in phase 2 of the learning cycle.
op.	Operator: AND, OR, XOR to indicate some actions happen either simultaneously, optionally, exclusively.
→	An arrow to indicate the flow to the next point.
action	Contains a phrase to indicate an action performed by a Knowledge Worker in phase 3 of the learning cycle.
Latest stage	Describes a latest stage of a flow sequence.
additional notes	Indicates additional notes.

Fig. 4. Guide map symbols

[4] The description of the CERN process [3] inspired the notation used for the CORONET-Train guide map.

The starting point of the CORONET-Train application guide map is shown in Fig. 5. It relates to the first phase of the learning cycle where the Knowledge Workers identify the learning need.

Fig. 6 shows a more detailed extract of the guide map related to problem-solving during project work. In this learning situation, as a first action, Knowledge Workers formulate the problem to be tackled. Then, they can choose from two cases, each case corresponding to a specific CORONET-Train learning scenario. The application guide map directs Knowledge Workers in choosing the most suitable case.

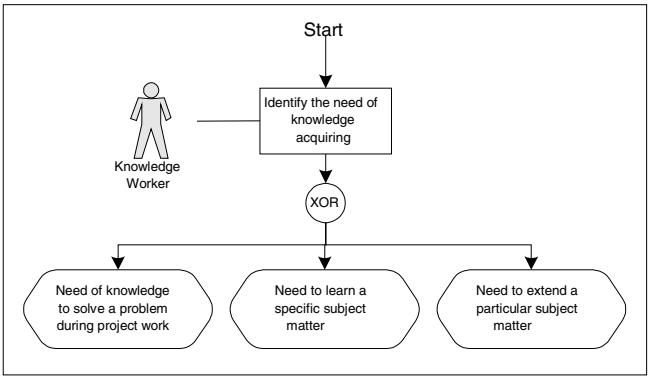

Fig. 5. Starting point of CORONET-Train application guide map

5 Application and Evaluation

At the industrial application partner sites of DaimlerChrysler and Highware, the CORONET system was evaluated in laboratory settings and under real-life conditions. Fig. 7 shows the differences between the evaluation contexts of the two companies (more details about the evaluation process and its results can be found in [7]).
The CORONET system was introduced and operated in both companies according to the processes presented in Section 4, adapted to the respective organizational perspective. One of the main evaluation goals focused on the effectiveness of learning with the CORONET system. The results of the evaluation studies are in its majority positive:

- The concepts contained in the learning methodology CORONET-Train are presented in a clear and concise style so that learners, trainers, tutors, and authors can easily identify the right learning scenario for their particular learning/training needs. This is particularly true for the scenario Web-Based Training.
- For training service provider Highware, the Virtual Classroom scenario offers a viable alternative to classical in-class training settings. The effectiveness of virtual classes was judged of being at least as effective as conventional in-class sessions. For the effective support of web-based experience sharing, the CORONET system successfully helped establish a network of geographically distributed learners and trainers.

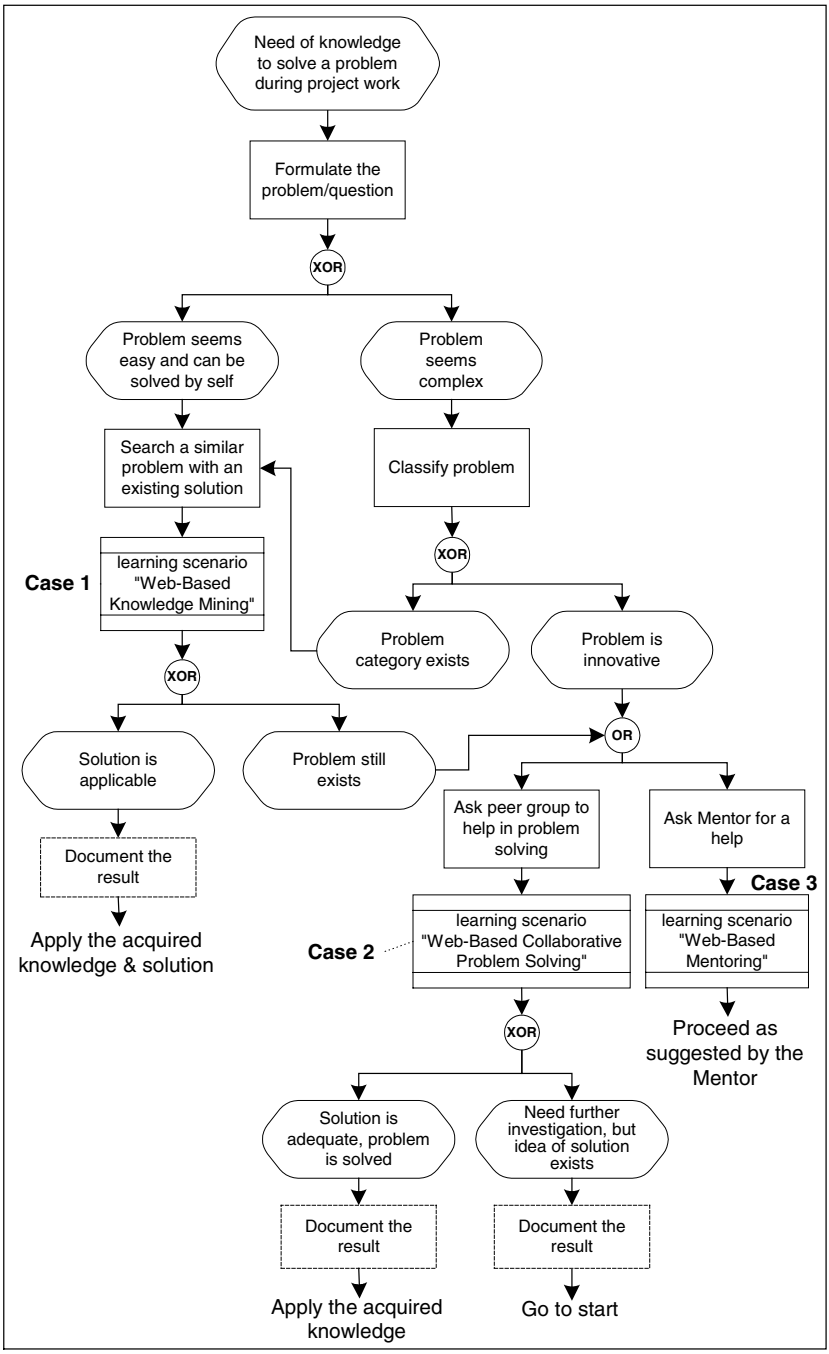

Fig. 6. Guide map extract related to problem-solving during project work

	Highware	DaimlerChrysler
Type of company	Training service provider	Industrial corporation
Number of evaluation users	32	40
Organizational culture	Small to medium-sized network-like organizational settings with strong service orientation	Complex team-oriented organizational setting with strong product focus
Evaluation approach	Kirkpatrick's scheme of evaluation [5]: measurement of effectiveness of training efforts	Cognitive-load theory [10]: think-aloud protocols during systematic observation
Learning scenarios evaluated	Web-based training Web-based virtual classroom Web-based experience sharing	Web-based knowledge mining Web-based collaborative problem solving Web-based mentoring

Fig. 7. Evaluation parameters

- The evaluation goals at DaimlerChrysler were led by the principle that the process of building up knowledge should be as efficient as possible and organized in a way that task-irrelevant cognitive load is as minimal as possible. The scenarios chosen were quite complex and would have demanded more time and effort for users to become acquainted with the system before starting any real learning activities.

Analyzing the results of the evaluation studies, with regard to the effectiveness of the CORONET-Train implementation and guidance processes, the results are not fully consistent. It clearly appeared from all evaluation studies in both companies that users need some time to handle the new learning environment before they can effectively focus on any specific learning activity. However, from the perspective of the training provider Highware, the available implementation and guidance processes seemed to be sufficiently useful for their learning scenarios, while the users at DaimlerChrysler seemed to be more sensitive to usability aspects of the software infrastructure and would have appreciated better guidance during performance.

Besides that, some observations and conclusions on cultural and organizational aspects can be drawn from the evaluation studies. The analysis of the CORONET system user profiles clearly showed that there was a positive predisposition to work with a web-based learning environment, as most of the users had been familiar with information and communication technologies for more than two years. This was certainly a positive influence factor for the introduction of CORONET-Train. On the other hand, some cultural factors were detected as being critical. Shifting to the successful operation of CORONET-Train requests changes in the behavior of nearly all the roles involved. The changes are mainly related to:

- the learning approach: shifting from the conventional presence learning mode to using the Internet is not obvious for learners who have not yet had experience with or have not been sufficiently prepared to using the new learning and knowledge transfer processes offered by the learning methodology;
- the pedagogical approach: replacing interpersonal relations typically occurring in conventional classroom settings by interactions between the learner and the web-based learning environment requires new competence on the part of trainers, tutors, and authors of learning materials.

These findings reinforce the assumption that for implementing a collaborative learning environment like the CORONET system, it is highly recommended to properly introduce both the methodology and the infrastructure to all types of users.

6 Conclusion and Future Work

The evaluation studies conducted during the CORONET project show that there are good chances to successfully introduce even complex new methodologies and learning infrastructures, provided that emphasis is placed on adequate implementation and guidance processes. Furthermore, the expectation was nurtured that by gradually discovering the wealth of collaborative learning and knowledge exchange according to CORONET-Train, users of the CORONET system will contribute to the improvement of the collaborative culture in their organization. Sustainability of learning effects and knowledge acquisition, however, still needs to be proven by long-term evaluation studies.

An interesting task for future work will be the extension of current CORONET-Train implementation processes towards reuse of WBT-Master patterns into company proprietary or commercial-off-the-shelf learning software infrastructures. Future work will also focus on improving the guidance processes of CORONET-Train in order to overcome the weaknesses in the guidance during operational use with complex learning scenarios as observed in the evaluation studies conducted by DaimlerChrysler.

Acknowledgements

The work presented was partly funded by the Information Societies Technology Programme of the European Union (IST-1999-11634). Special thanks go to Prof. Nick Scerbakov's team at the Technical University of Graz for developing the learning infrastructure WBT-Master, and to DaimlerChrysler and Highware for conducting evaluation case studies in industrial settings. More information on the CORONET project is available at URL http://www.iese.fraunhofer.de/coronet.

References

1. V.R. Basili, G. Caldiera, H.D. Rombach, "Experience Factory", in: Marciniak JJ (ed.), Encyclopedia of Software Engineering, Vol. 1, pp. 469-476, John Wiley & Sons, 1994.
2. A.J. DiBella, E.C. Nevis: How Organizations Learn – An Integrated Strategy for Building Learning Capability, Jossey-Bass Publishers, San Francisco (1998)
3. M. Haidt, Knowledge Creation Process in CERN. White paper, University of Tübingen, available at URL http://www.haidt.de (1999)
4. D. Helic, J. Lennon, H. Maurer, N. Scerbakov: Aspects of a Modern WBT System. In: SSGRR 2001, Loughborough University (2001)
5. D. L. Kirkpatrick: Evaluating training programs – The four levels, Berrett-Koehler Publishers, San Francisco (1998)
6. D. Pfahl, N. Angkasaputra, C. Differding, G. Ruhe: CORONET-Train: A Methodology for Web-Based Collaborative Learning in Software Organizations. In: Proceedings of the 3rd workshop on Learning Software Organizations, Kaiserslautern, Germany (2001)
7. D. Pfahl, S. Trapp, et al.: "CORONET Final Report", available from the CORONET project web-page at URL http://www.iese.fraunhofer.de/coronet (2002)
8. G. Ruhe, F. Bomarius (eds.): Learning Software Organization - Methodology and Applications, Springer-Verlag, Lecture Notes in Computer Science, Volume 1756 (2000)
9. E. H. Schein: "Organizational Culture and Leadership", San Francisco, CA: Jossey-Bass, 1992.
10. J. Sweller: Cognitive-load theory, learning difficulty and instructional design, pp. 295-312. In: J. Sweller: Learning and Instruction (1994)
11. WBT-Master User Manual, available at URL http://coronet.iicm.edu/manual/manual.zip (2002)

Building Communities among Software Engineers: The ViSEK Approach to Intra- and Inter-organizational Learning

Britta Hofmann and Volker Wulf

Fraunhofer Institute for Applied Information Technology (FhG-FIT)
Schloss Birlinghoven, 53754 Sankt Augustin, Germany

Abstract. The paper presents the concepts of the Distributed Center of Competency in Software-Engineering (ViSEK): a German national initiative to encourage intra- and inter-organizational learning in the software industry. Taking a socio-cultural stance, learning is understood as enculturation in a community of practice. So community building becomes an important objective when fostering intra- and inter-organizational learning. The ViSEK-project encourages community building among software-engineers at three different levels: between leading research groups, between research groups and practitioners, and among practitioners from different small and medium size enterprises (SMEs). We describe two approaches for community building more in detail: (a) an internet portal which presents software engineering knowledge and offers functionality for community support, (b) a regional network of SMEs which stimulates learning among its members in the field of usability engineering.

1 Introduction

The field of Software-Engineering develops rapidly. Furthermore software development knowledge is very specialized and fast out-of-date. Especially for small and middle-sized enterprises (SMEs) it is hard to keep pace with actual software trends and the requirements of dynamically changing markets. SMEs typically have neither own divisions for personal development nor enough time and financial resources to visit expensive and time-consuming seminars. To be in business with big software competitors SME have to meet two challenges: First they have to stay up-to-date with theoretical software engineering knowledge; second they need practical experiences in state of the art technologies and methods to decide if it is worth to learn a new engineering method or to pay for a development tool. In other words: SMEs need expert knowledge and practitioner experience to fit the requirements of their markets.

Due to the fact that SMEs play a major role in the German software industry (Broy et al. 2001), the Federal Ministry of Education and Research funded the ViSEK Consortia to support German SMEs in coping with these two challenges. ViSEK is the acronym for "Virtual Software-Engineering Competence-Center". ViSEK has two main goals. First, it should consolidate the state of the art in software- engineering and develop an internet portal which presents practical relevant knowledge in a repository structure (cf. Feldmann and Pizka 2002). Second, it should build up a lively software engineering community that brings together SMEs from the software and media and industries with researchers from the field of Software Engineering.

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 25–33, 2003.

Effective learning and knowledge creation does not take place in social isolation.

In fact, learning is a constructive and socio-cultural process (see Bateson 1983; Lave and Wenger 1991; Wenger 1998). Therefore, the ViSEK portal offers besides the knowledge repository options to discuss and exchange knowledge. These community features enable SMEs not only to collaborate with researchers but also with colleagues from other software companies. Such a conjunction between research institutes and SMEs holds mutual benefit. On the one hand SME practitioners gain insights into state of the art of software engineering knowledge. On the other hand researchers get feedback to focus their work towards challenges which are relevant in practice. The exchange and collaboration of different SME triggers inter-organizational learning effects. This is very important, since SMEs have to focus on their core competences. Learning should therefore include aspects of network building to cope with complex problems and projects.

2 Learning as Enculturation in Communities

ViSEK as a learning platform stands on two pillars: software engineering repository and community. We believe that according to recent learning studies pure repository approaches grasp to short in supporting people to learn effective and efficiently (Davenport and Prusak 1998; Ackerman, Pipek, and Wulf 2003).

Pure repository approaches are based on an "instructionist" understanding of learning. The learner is seen as receptive system, which stores, recalls and transfers knowledge. This understanding was criticized from theoretical and practical point of view (see Collins et al. 1989, Schulmeister 1997). Thus recent scientific approaches favor constructionist and socio-cultural concepts of learning. Based on the work of Piaget and Bateson (1983) learning is seen as an active and constructive process (cf. Prenzel & Mandl 1993). Learning means not just transferring knowledge. Learning is rather the permanent construction of knowledge, based on former experiences. Hence knowledge is linked to real world problems.

In the last decade constructionist theories of learning played an important role in the development of new computer-based learning-forms (Duffy und Jonassen 1992, Gräsel et al. 1997).

Socio-cultural learning theories refer to Vygotsky (1962). They are based on ethnographical studies in different cultures (Lave und Wenger 1991). Those theories take learning as a collective process and linked to specific contexts of action. Knowledge emerges in communities of practice by discursive assignment of sense. Communities of practice are characterized by common conventions, language, tool usage, values and standards. Learning is understood as the gradual inclusion or insertion into a community of practice (enculturation).Enculturation is consolidates by "cognitive apprenticeship" (Collins et al. 1989, Lave and Wenger 1991; Wenger 1998).

3 Software Engineering in Germany

The current situation in the German software industry is characterized by lacking social networks especially among different SMEs and lack of exchange between software engineering practitioners and researchers. Looking at the SE-research insti-

tutions, we find quite some groups of excellent researchers, spread over Germany. However, these groups are poorly connected; especially what concerns the operational level. They focus on different topics, following different research agendas and paradigms. Moreover, innovative highly interdisciplinary fields such as usability engineering are not well enough connected with the traditional SE groups.

We believe that diversity is the motive power of scientific progress. However, diversity only contributes to scientific advance, if there are explicit links and discourses among the different research groups. This requires mutual understanding and the development of shared concepts. Consistent software engineering concepts are so far out of sight. A consistent terminology is still missing. Same terms (e.g. "interface" or "component") mean in different sub areas completely different things; in other cases different terms have the same meaning. For theoretical and practical reasons, we do not believe that a unified language is achievable for a heterogeneous research field such as Software-Engineering. However, we believe that a more intense scientific discourse between the different subfields would lead to new insights. Moreover, an unreflected use of terms leads to particular confusion in collaboration with practitioners.

Looking at the relation between SMEs in the software and media industry, we find a lacking culture of trust and cooperation. This may be explained by their fear of competition. However, without inter-organizational cooperation SMEs have severe problems to handle big and complex projects, to keep pace with software trends, and to follow requirements of dynamic markets.

Therefore, ViSEK faces three main challenges: The first challenge is to strengthen the German scientific software-engineering community. The second challenge is to trigger cooperation among the SMEs of the software and media industry, and the third challenge is to bridge between sciences and practice.

4 The Approach Taken by ViSEK

The scientific consortium of ViSEK consists of eight research institutes: five Fraunhofer Institutes and three groups based at research universities[1]. The different knots of the network cover the different regions of Germany rather well.

To prepare for community building on the national level, all partners work together in organizing press conferences or other public relations activities to communicate the need for cooperation among SMEs and IT-research.

More importantly, the joint design of the web-based portal triggers community building among the scientific institutes. While the implementation of the portal is carried out by one of the partners based on an existing content management system, the scientific consortium has to cooperate closely in the design of the portal. Designing a SME-oriented presentation of software-engineering knowledge is a complex process with lots of interrelated cooperative activities. Here we will focus on three activities the scientific consortia is concerned with: (a) creating a joint classification scheme for software engineering knowledge as a basis for the main navigation structure of the portal, (b) establishing a common terminology as a basis for a joint index,

[1] Fraunhofer IESE, Kaiserslautern; Fraunhofer FIT, Sankt Augustin; Fraunhofer FIRST, Berlin; Fraunhofer IITB, Karlsruhe; Fraunhofer ISST, Berlin and Dortmund; Technical University of Munich; OFFIS at the University of Oldenburg; Technical University of Cottbus.

(c) reviewing each others content for the repository to ensure quality standards and readability. All these activities support community building effects among the scientific institutions, since they are forced to refer to each others work in achieving the common goal.

The scientific partners agreed to apply the SWEBOK 0.95 classification scheme of software engineering knowledge to be the starting point for creating a navigation structure inside the portal. So each partner started classifying his specific knowledge according to the SWEBOK scheme. This activity leads to intense discussions among the partners, since they had to refer to each others contributions as well as to agree on modifications of the SWEBOK classification scheme. The creation of a classification scheme for navigation is a highly political activity because it influences the accessibility of the different partners content.

As we assume that SMEs produce software for specific application domains, we designed two domain-specific access modes for the portal: E-Business Applications and (Safty) Critical Systems. The different partners have to integrate their content into this domain-specific structure where possible. This way they have to discuss and agree on a domain-specific coverage of the field, as well.

Another challenge is the elaboration of a knowledge index. In the different schools and subfields of software-engineering, there exist various meanings of the same terms. How to deal with this problem? One possible solution would be to use only a high level definition of these terms. The disadvantage of high level definitions is the fact that these definitions are not very meaningful to users, especially from SMEs. So the consortium agreed that, in specific cases, it would be better to allow different well-defined concepts in parallel. The joint reflections during the portal building process triggers mutual learning among the scientific research communities and connects the individual German research institutes more closely.

Additionally to German-wide community activities the scientific partners organize community building activities on the local level. This is very important since we cannot hope to create a SE-community on the national level in a first step. People exchange knowledge only in case they trust each other. It is very helpful in establishing confidence, when people can easily meet each other and discuss, without having to bridge big distances. As a first step we decided not to build a pure online-community. We use the web-based portal as a medium to organize real world communities, which may meet in the virtual space, as well.

In the following we exemplify the local community building activities undertaken by one of the ViSEK partners: the Fraunhofer FIT in the Bonn region. Similar activities are initiated by other ViSEK partners.

4.1 Building Local Communities

The Bonn region is characterized by a high density of SMEs in the software and media industry. Many of them provide services or market products to the local telecommunications, television, and insurance industries. Recently the research group at Fraunhofer FIT has organized a series of social events for regional SMEs from the software and media industry. The target groups of these events were managers and software developers.

A goal was to make the ViSEK initiative known and to involve the local SMEs into the design of the web-based portal in which practically proven software engineer-

ing knowledge will become accessible (see section 4.2). We also intended to foster inter-organizational learning and cooperation among local SMEs. Finally, we wanted to strengthen our ties with industries to enable knowledge transfer into practice and to align our research efforts more closely with industry needs.

These social events were organized in form of topic centered workshops and focused on the subfield of usability engineering. Why did we focus on only one topic and why did we choose usability engineering out of different other subfields of software engineering our institute works in?

We assumed that setting a focus leads to more tangibility and thus to more attractiveness. If we had invited the SMEs to discuss software engineering in general, it would be more difficult to communicate the usefulness of the events. We assumed that SME do not want to spend their time to talk globally without a predefined goal. The goal we defined was to encourage learning in a specified niche to increase competitiveness in the market. We chose "usability engineering" since this topic demands interdisciplinary team work. Moreover, usability activities are normally not well established in software SMEs in these days. This makes learning effects easy measurable. One indicator for the learning success may be the augmentation of acceptance and the growth of a positive attitude towards usability. At the same time many software SME refuse to spend money and time to build up usability departments in their enterprises, since they cannot easily evaluate the economic outcome.

For each of the three workshops all regional SMEs were invited by means of a letter sent to their management. More than 500 letters were distributed to a list of companies provided by the local chamber of commerce. At each workshop we attracted between 30 and 45 participants. There was a core of about five to seven companies which participated in all of the events. Besides this, we had a rather big group of actors who only took part in one or two of the events.

The procedure of the workshops was as follows: First researches gave talks about usability relevant issues. The auditorium had the possibility to ask questions and to discuss these issues with the speaker right after the talk. We found a lively interest in discussion and collaboration with researchers. For SMEs it seems to be very attractive to promote their products and services by demonstrating cooperation with well-known research institutes. However, our experiences so far indicate that SMEs are more interested in "decorating" scientific contacts than in the actual state of the art in research. So we have to work out strategies to augment their interest in the topic itself.

After the talks and following discussions we invited for a reception where the SMEs and the IT-researchers could network in a more informal way. Concerning the network-building among SMEs, we noticed an interesting phenomenon: when we first confronted them with the idea of learning from each other and cooperating with each others, they reacted negatively. They argued that they are in mutual competition and that it would be short witted to give away their in-house knowledge that ensures their advantages in the market. But when the managers talked to each other in small groups of three or four people, they became aware that they suffer from similar problems. Under these circumstances they were easily willing to exchange information and advice. From a psychological point of view we learned that for community building it is essential to create a joint awareness of common problems, so that the SMEs feel that there is a common goal to achieve. As a next step, we involved the management of this core group in founding an association which is supposed to become the institutional base for the ongoing regional networking activities.

We will draw on this insight gained from building local communities when trying to build online-communities in the portal. Additionally other core prerequisites for multidisciplinary exchange and cooperation between SMEs must be taken into account and be realized by ViSEK portal. Whittaker et al. (1997, p. 137) identified the following core attributes:

1. Members have a shared goal, interest, need, or activity that provides the primary reason for belonging to the community.
2. Members engage in repeated, active participation; often, intense interactions, strong emotional ties, and shared activities occur among participations.
3. Members have access to shared resources, and policies determine the access to those resources.
4. Reciprocity of information, support, and services among members is important.
5. There is a shared context of social conventions, language, and protocols.

In combination with the knowledge repository of the ViSEK portal FIT embraces to this attributes by organizing regularly social activities in "real world". The challenge is the transfer of these attributes in the virtual world of the ViSEK portal.

4.2 Building a Community-Centered Portal

The ViSEK portal is based on two pillars: a knowledge repository and online communities. The goal of the repository structure is to provide an easy and fast access to the most actual software engineering knowledge. In this way it creates an interesting place in the virtual world which attracts attention by people interested in innovations in software-engineering methods, tools and processes. So far the eight scientific partner organizations made their software engineering knowledge available to the SMEs by creating content for the knowledge repository (cf. Feldmann and Pizka 2002). While it may create an interesting space in the virtual world, a repository by itself does not support the emergence of communities. Hence additional efforts must be undertaken: functionality for community-support needs to be included.

To realize the portal in a resource-efficient manner, we chose the content-management system *WebGenesis* which was provided by one of the scientific partners (Fraunhofer IITB). This system offered extensive functionality to input, edit, structure and display the knowledge repository. Moreover, it contained some community-oriented functionality, such as discussion groups. However, the basic functionality of the content management system does not yet provide a tide enough integration of repository and community-building functionality. For instance, we would like to allow users to annotate certain aspects of the content which then should be displayed to the other users. This way discourses with regard to the content of the portal could be supported. With regard to functionality for community-support, we were forced to initiate a software development process to extent the given platform.

Community-centered development focuses on the communities needs prior to making decisions about the technology (cf. Preece 2001). With regard to user-centered interaction design, different techniques are already worked out (e.g. Preece and Rombach 1994; Kreitzberg, 1998; Shneiderman, 1998a). In accordance with HCI approaches, we identified first the needs of SMEs from the software and media industries for learning and knowledge exchange. We created more than 40 textual scenarios for the portal's use and an early prototype to present our vision to representatives of

software SMEs primarily taken from our local network. Based on this feedback we derived requirements for the design of the ViSEK portal, especially the community-supporting functions.

We realized an early version of the portal based on the content management system. This version contained only limited community-oriented functionality. It was presented on a workshop of the local network and on the CeBIT 2002 (a major trade-show for information and telecommunications technology) to gain additional feedback from our target group. By means of the user-centered design process, we identified the following needs for community support in the portal:

- ✓ *Exchange practical experience*

  It is one thing to read something about a new method or a new technology. The other thing is to establish this method or technology in your own enterprise. The implementation of new technologies normally comes along with significant changes in the whole organization. Changes often cause diminished turnovers in the first step. Therefore it is useful to be preliminary informed about possible disadvantages at first hand. So the ViSEK portal should not only offer scientific information about innovative software engineering methods and tools. Additionally practitioners should have the possibility to annotate these methods by reporting about their own practical experiences. Different experiences could be related towards each other. So practitioners are able to discuss among themselves the pros and cons of to the deployment of an innovative method, tool or process in their software engineering practice.

- ✓ *Discuss in an asynchronous mode*

  One can distinguish between two modes of computer-mediated communication: synchronous and asynchronous. Synchronous information exchange means communication in real time. Participants have to be present at the same time, though not necessarily in the same place, so that comments can be followed up instantly within the communication. One realization of synchronous discussions would be chat-rooms. Asynchronous conversation does not require participants to be available at the same time. For the ViSEK portal we decided to provide primarily asynchronous communication channels among the SMEs, because almost none of the SME employees we interviewed could imagine chatting in real time with other colleagues during a normal working day. We favor establishing electronic discussion groups. We still have to analyze which topics to choose for initial discussion rounds.

- ✓ *Refer to experts*

  In knowledge management it is a well known fact that certain types of knowledge cannot be expressed verbally (so called tacid knowledge, according to Polanyi,1983). Users in the portal may not express their knowledge textually due to the efforts involved. Some experts may also hesitate expressing their knowledge fully in a portal because they would like to market it (cf. Davenport and Prusak 1998; Ackerman, Pipek, and Wulf 2003). Implicit knowledge is definitely needed with regard to the moderation of organizational change in software-engineering processes. Therefore, a community-based portal needs to provide reference to experts. The portal will offer expert profiles (“yellow pages”) with contact information so that SMEs are able to find the fitting expert for their problems. The experts' yellow pages will also be linked to those parts of the repository their

expertise refers to. Moreover, we will implement functions which match actors with similar interests and learners with experts via appropriate algorithms and strategies for visualization (cf. Becks, Reichling, and Wulf 2003). In this way the portal can serve as a platform to match experts with actors who look for a solution to their problems. Additionally ViSEK experts and famous software engineering professionals will be available in chat rooms at pre-defined points in time. Representatives of SMEs stated, that they are interested in synchronous communication with experts in case it does not happen too frequently and does not take too much of their time. In contrast they preferred asynchronous exchange with colleagues.

5 Conclusions

Socio-cultural theories stress the importance of communities of practice in the process of learning. Seen from this perspective, repository approaches which are based on the assumption that relevant knowledge can be explicated, represented, and diffused via electronic networks are problematic (cf. Basili et al. 1994). They need to be augmented by approaches which foster community building. In the software industry, community building will always require an integration of technical and social activities. These findings have to be always taken into account when reflecting on learning software organizations.

Based on our theoretical background, we have described approaches to community building within the German software industry. The current situation in the German software industry is characterized by insufficient networks among researchers, between practitioners and researchers, and among practitioners. The ViSEK project aims to establish a culture of inter- and intra-group cooperation by means of an internet portal and local social events. The internet portal offers state of the art software engineering knowledge and provides functionality to build (online) SE- communities.

We are in the midst of a complex process with challenges on the social as well as on the technical side. In this paper we have presented our conception and some early results. Yet it is unclear whether and to which extend this conception will turn out to be successful. The different social and technical measures for community building need to be evaluated carefully. Moreover, one has to investigate how to treat online-communities in the framework of socio-cultural theories of learning.

One of the biggest practical challenges is to establishing a culture of trust and cooperation among SMEs, and especially among their managers. Right now a competitive culture seems to dominate their relationships, which hinders knowledge exchange, mutual learning, and inter-organizational cooperation.

Acknowledgements

We would like to thank our colleagues Helmut Simm, Andrea Bernards, Doris Kamnitz-Kraft, and Bernhard Nett for supporting our community building activities in a variety of different ways. Ralf Kalmar (FhG-IESE) provided insightful comments on an earlier version of this paper. The ViSEK-Consortia is funded by the German Ministry of Education and Research (01 IS A02).

References

1. Ackerman, M.; Pipek, V.; Wulf, V. (eds). Expertise Sharing: Beyond Knowledge Management, MIT-Press, Cambridge MA 2003
2. Basili, V. R., Caldiera G., Rombach, H. D. (1994). *Experience Factory,* in: Encyclopedia of Software Engineering, Vol 1, p. 469-476.
3. Bateson, G (1983). *Ökologie des Geistes*. Frankfurt: Suhrkamp
4. Becks, A.; Reichling, T.; Wulf, V.: Expertise Finding: Approaches to Foster Social Capital, in: Huysman, M.; Wulf, V. (eds). Social Capital and the Role of Information Technology, Kluwer, Dordrecht 2003, in preparation
5. Broy, M.; Hartkopf, S.; Kohler, K.; Rombach, D.: Germany: Combining Software and Application Competencies, in: IEEE Software, 18 (4), July/August 2001
6. Collins, A., J. S. Brown und S. E. Newman: Cognitive Apprenticeship: Teaching the Crafts of Reading, Writing and Mathematics, in: L. B. Resnick (ed.). *Knowing, Learning, and Instruction*, Hillsdale: Lawrence Erlbaum Associates, 1989, pp. 453 – 494.
7. Davenport, T.H. and Prusak, L. (1998): *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press, Boston, MA, USA.
8. Duffy, T. M. und D. H. Jonassen (eds): *Constructivism and the Technology of Instruction: A Conversation.* Hillsdale: Lawrence Erlbaum Associates, 1992
9. Feldmann, R. L.; Pizka, M.: An On-Line SE-Repository for Germany's SMEs: An Experience Report, submitted to: International Workshop on Learning Software-Organizations (LSO 2002)
10. Gräsel, C., J. Bruhn, H. Mandl und F. Fischer. „Lernen mit Computernetzwerken aus konstruktuvistischer Perspektive.“ *Unterrichtswissenschaft* 25, Nr. 1 (1997): 4–18.
11. Kreitzberg, C. (1998). *The LUCID Design Framework (Logical User-Centered Interaction Design).* Princeton, NJ: Cognetics Corporation.
12. Lave, J. und E. Wenger. *Situated Learning: Legitimate Peripheral Participation.* Cambridge University Press, 1991.
13. Polanyi, M. (1983). The Tacit Dimension. Magnolia, MA: Peter Smith Publishers. ISBN: 0-844-65999-1.
14. Preece, J. & Rombach, D. (1994). A taxonomy for combining software engineering (SE) and human-computer interaction (HCI) measurement approaches: towards a common framework. *The international Journal of Man-Machine Studies*, 14, 553-583.
15. Prenzel, M. und H. Mandl: „Transfer of Learning from a Constructivist Perspective.“, in: T. M. Duffy, J. Lowyck und D. H. Jonassen (eds): *Designing Environments for Constructive Learning* Berlin, Heidelberg: Springer, 1993, pp. 315 – 329
16. Preece, J. et al (2001). *Interaction Design*. New York, NY: John & Wiley & Sons, Inc.
17. Schulmeister, R. *Grundlagen hypermedialer Lernsysteme: Theorie - Didaktik - Design.* 2. Edition, Munich, Oldenbourg, 1997
18. Shneiderman, B. (1998a). *Designing the User Interface: Strategies for Effective Human Computer Interaction (Third Edition).* Reading, MA: Addison-Wesley..
19. SWEBOK 0.95: http://www.swebok.org
20. Vygotsky, L. S.: *Thought and Language.* Cambridge: MIT Press, 1962
21. Wenger, E.: *Communities of Practice : Learning, Meaning, and Identity.* Cambridge University Press, 1998.
22. Whittaker, S., Issacs, E. & O’Day, V. (1997). Widening the net. Workshop report on the theory and practice of physical and network communities. *SIGCHI Bulletin*, 29(3), pp. 27-30.

An On-Line Software Engineering Repository for Germany's SME –An Experience Report–

Raimund L. Feldmann[1] and Markus Pizka[2]

[1] Kaiserslautern University of Technology, AG Software Engineering, Postfach 3049, D-67653 Kaiserslautern, Germany
r.feldmann@computer.org
[2] Technische Universität München, Institut für Informatik – H5, D-80290 München, Germany
pizka@in.tum.de

Abstract. In order to keep pace with competitors in the ever accelerating business world, software organizations have to continuously improve their products and processes. However, SMEs typically do not have the time to invest in costly training programs and test the newest technology on their own. Most of the time they need support for their daily work processes, which often employ agile methods. In this paper, we exemplify how SE repositories can be employed to support such settings. A taxonomy for the content of such SE repository systems is given. Based on this taxonomy, we describe the internal repository structure of Germany's ViSEK portal: A portal implemented for offering up-to-date SE knowledge to support SMEs in their daily work.

1 Introduction

According to a recent study on Germany's software industry [3], many companies use and develop software to improve their products and processes. This includes large companies, such as car manufactures, as well as about 20,000 small and middle-sized enterprises (SMEs). Common to most of them is the fact that traditionally, these companies have not focused on software development. Hence, they often lack experience and need support for their daily work processes. The same holds for some of the software houses that mainly focus on software development.

While larger companies and software houses may have enough time and money to hire experts, invest in costly training programs, and/or test the newest technology on their own, this is usually not true for SMEs. For the mentioned reasons the need for specific Software Engineering technology support of Germany's industry—especially the large number of SMEs—is seen as a key issue for our economy. Therefore, the Department of Education and Research (bmb+f) of the German Federal Government funded the ViSEK project [12]. Based on the idea that experience gained from research and practice should be packaged and easily accessible to all companies, an on-line SE repository is being installed in the course of the ViSEK project. Similar to other repository systems (e.g., [8], [10]), the ViSEK portal offers access to up-to-date

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 34–43, 2003.

Software Engineering technologies of selected application domains. This should help to improve best practices and support daily work, independent of weather a company uses traditional software development methods, or agile methods.

The idea of a German, web-based Software Engineering (SE) repository is similar to the NFS (National Science Foundation) supported CeBase project [4] in the US. As with CeBase, the basic idea behind the ViSEK portal is the Experience Factory concept, as suggested by Basili et. al. [2], with its SE repository, the Experience Base. However, while CeBase mainly focuses on supporting people from the areas of research and education, ViSEK's main target group are SMEs that develop software on a daily basis.

In this paper we present the current structure of the internal ViSEK repository, which is the result of several discussions and iterations. The starting points for its definition were the process pattern approach developed by Gnatz et. al. [5] and the repository structure architectural framework developed at the University of Kaiserslautern [6]. The remainder of this paper is organized as follows: Section 1 discusses the question of how SE repositories like the ViSEK portal are able to support SMEs and their agile processes. Then, in Section 2, we give a taxonomy for the content of such SE repositories in general, before we describe the content and storage structure of the ViSEK portal in detail (Section 3). Finally, we summarize our experiences and conclude with future directions in Section 4.

2 Motivation

ViSEK's goal to deliver SE expertise to SMEs creates unique challenges. SMEs are usually not able to invest in costly training or consulting activities. Even if trainings are offered at low cost, our experience shows that SMEs often are still not willing to release their employees from work for training purposes. Hence, to be successful with the ViSEK SE portal in improving SE practices in SMEs, we have to understand and respect the specific constraints and needs of SMEs:

1. SMEs are not able to make significant investments into any a priori learning activities without concrete needs. E.g., the temporary loss of a single developer of a company with 10 employees means immediate loss of >10% manpower for production.
2. SMEs need effective ways to quickly disseminate knowledge and experiences among project members on demand, because they are not able to afford groups of experts that can be deployed occasionally.
3. Nevertheless, there is a strong need and demand for long-term learning, because software SMEs operate in a very competitive market where skill is crucial for long-term success and survival.
4. SMEs must often cope with particularly tight schedules and limited budgets while constantly dealing with new challenges. As a consequence they are often in need of ad hoc (~ instantly created for the purpose at hand) methods and solutions. SMEs are particularly interested in acquiring more knowledge related to agile software development.

Thus, long-term learning in SMEs is mandatory but must be performed in a delicate balance with the agility of SMEs and their processes. The information needs of SMEs must themselves be considered as "agile". Information must be accessible with low

overhead and concrete answers to questions in certain problem contexts must be readily available. The ViSEK repository scheme aims at supporting this agile style of information delivery by a light-weight repository structure (Section 4), the careful selection and dedicated preparation of information made available (problems, technologies, experiences), and an adaptive, web-based, and low-cost user-interface.

Usage Scenarios

Based on our observations concerning the particular situation in SMEs, our strategy to support learning in SMEs is based on the following major usage scenarios for the ViSEK Internet portal:

1. A project team member quickly needs information concerning a specific technology to perform a task he or she has not performed before, such as estimating costs with a certain model. For this purpose, the ViSEK repository provides brief technical descriptions of important SE techniques that support that task.
2. A project team or member is facing a concrete SE problem, such as a high defect rate, and is looking for possible solutions. To quickly aid in this common situation, the ViSEK repository provides descriptions of known problems with references to experience reports and possible technical solutions. By matching the concrete problem situation with the problem descriptions provided by ViSEK, the user has a chance to quickly identify similar problems and possible solutions.
3. To strengthen the position of his organization in the market a SME manager wants to improve aspects of the SE practices of the organization but does not know the economic impact of different technologies, such as inspections. ViSEK supports SME managers by providing experience reports related with technologies, because SMEs can not afford to try new technologies on their own.

In addition to these scenarios, SMEs are often forced into some kind of ad hoc software development, as stated above. Some of them have already heard about agile methods but do not know "how to do" XP or other agile techniques and "when [not] to do" it. The ViSEK portal will provide answers to these questions by providing SME experience reports, while relying on links to other web sites for descriptions of technical details where possible.

3 A Taxonomy for the Content of SE Repositories

As indicated by the scenarios, SE repositories can be employed in different ways to support the daily work of SMEs. To describe the content of such SE repositories more precisely, we make use of the terms *data*, *information*, *knowledge*, and *experience* (see Fig. 1). For our taxonomy, we adopt the definitions of these terms as provided by Aamodt & Nygard [1] and Tautz [11]:

- ***Definition 1: Data*** are patterns with no meaning; they are input to an interpretation process, i.e., to the initial step of decision making[1].

[1] Note that this definition is also similar to the one given by Lehner [9], who describes data as "*raw, unsummarized, and unanalyzed facts*".

- ***Definition 2: Information*** is data with meaning that a human or an intelligent information system assigns to this data by means of conventions used in their representation[2].
- ***Definition 3: Knowledge*** is the range of learned information or understanding of a human or an intelligent information system.
- ***Definition 4: Experience*** is gained by humans through utilization of information or knowledge.

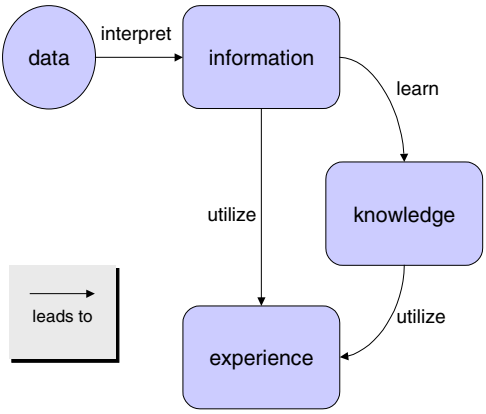

Fig. 1. Relation between data, information, knowledge, and experience

According to Definition 1, all artifacts that occur during the development process of software can be denoted as data, regardless of their quality, quantity, or representation. Examples for data are: (undocumented) code, test cases (i.e., test *data*), requirements documents, or collected measures of effort, errors, or complexity (i.e., measurement *data*). All project related artifacts (i.e., the data) are typically stored in project databases. Since the stored project data are the starting point for creating information, knowledge, or experience by interpretation, learning, or utilization, project databases can be regarded as an integral part of SE repositories.

Creating a project database by following a certain storage structure (i.e., a convention for the data representation) is the first step towards transforming the stored data into information. With a well-defined storage structure, it is possible to automatically create information. Effort distribution models according to the project phases requirements engineering, design, coding, and testing, for instance, can be automatically accumulated from the collected measurement data stored in the project databases. However, information can not always be created automatically. Other information is created by human interactions and has to be integrated into the SE repository manually. Examples for such types of information are: carefully documented C++ or JAVA classes, process and product models gained from former projects that are completely stored in a SE repository, or a collection of (standard) test cases. The common denominator of all this information is the fact that it usually can not and should not be stored as part of the project databases. Hence, storage structures for information form another part of SE repositories.

As indicated by Definition 3, knowledge is gained by a learning process based on information. Learning, however, is always an individual (i.e., subjective) process for a human or an intelligent information system, which depends on the actual environment. Based on this environmental setting, the input information is processed and interpreted. Hence, it is important to record data and/or information that describes this

[2] Lehner defines information in [9] as "*data processed into a meaningful form*" and adds that "*one person's information can be another's data*". This stresses that the interpretation process from data to information is an individual process depending on the actual human or intelligent information system.

environmental setting (i.e., the context) in which the knowledge has been gained. Otherwise, knowledge is not explicit (i.e., applicable) for others in a similar context, and therefore, would have to be regarded as information again. A process model, for instance, can only be effectively (re-)used when the environment (e.g., the application domain, number of persons in project team, etc.) in which it was already successfully used is *known*. Since we want to avoid mistakes and use the learned knowledge that others have gained before, storage structures for knowledge should be an integral part of SE repositories.

Finally, Definition 4 indicates that the differences between knowledge and experience are:

1. that only humans can gain experience (not an intelligent system), and
2. the fact that experience is utilized.

The second point stresses that one explicitly makes use of the given information or knowledge and *experiences* the results (i.e., one does not only have to believe what others learned or interpreted). According to [11] the validity of experience is bound to the context of the event or activity in which it was gained. This implies that the context in which information or knowledge was utilized has to be stored with the experience, the reason being the fact that when experience is documented, it becomes knowledge for all others who have not experienced it on their own. However, in an SE repository it is possible to differentiate between knowledge and experience. While it is sufficient for knowledge to offer storage structures for the context, the storage structures for experience also have to include the complete environment (i.e., project) where it was gained in.

4 The Internal Repository Schema

With the help of the taxonomy introduced in the previous section, we now describe the main elements of the current storage structure of the ViSEK portal [12]. Our basic considerations for the repository schema design can be summarized as follows:

C1: **No data:** Storing pure data would not help ViSEK users in solving their daily problems. This is because they would not be able to compare weather the specific item is applicable in their setting (i.e., problem domain) or not. Since the ViSEK consortium does not conduct development projects on its own, it is also not necessary to store complete project databases in the ViSEK portal.

C2: **Knowledge as primary content:** To stress the initial ViSEK goal of providing up-to-date and high quality SE knowledge and allowing goal-oriented reuse, we insist on a precise context description for all stored technologies.

C3: **Only selected information:** To provide practical usage examples of described technologies in the ViSEK portal, we allow the storage of *project descriptions* in a compact form. These project descriptions capture the basic information about the projects (i.e., consist of accumulated data). Information is further accepted in the form of:

- *Glossaries* (i.e., basic definitions independent from a certain context),
- *Literature references* (i.e., standard descriptions of technologies like, for instance, UML), or

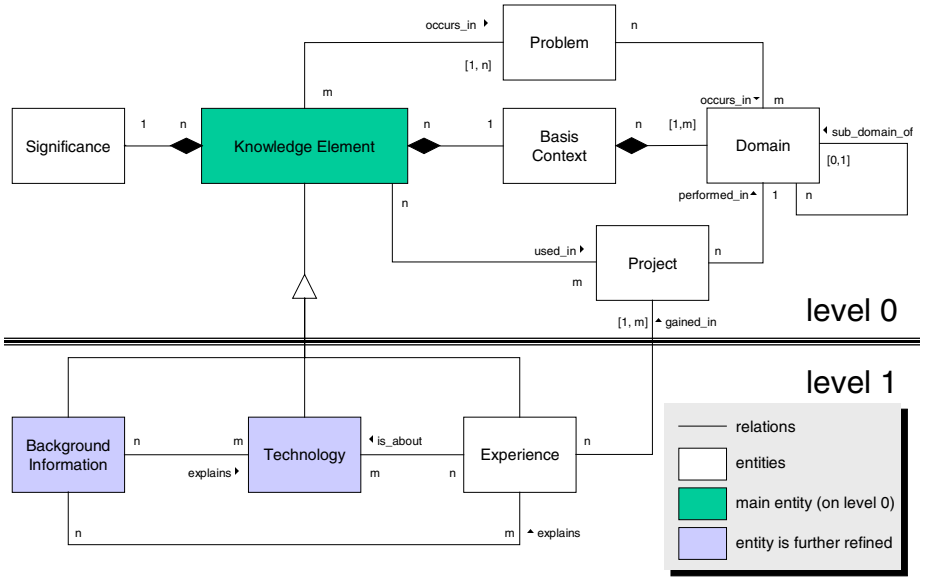

Fig. 2. ViSEK's internal repository schema: levels 0 and 1

- *Contact & expert information* (i.e., similar to literature references, humans or institutions are listed that can be refered to if specific questions regarding a certain technology arise).

However, Information that can not be directly related to any of the stored knowledge in the ViSEK portal is not to be stored at all.

C4: **Experiences are represented as lessons learned:** Experiences gained by ViSEK technology providers (i.e., the initial ViSEK consortium or registered users of the ViSEK portal) is documented in the form of lessons learned (i.e., knowledge) that include at least one project description (see C3 for details).

In addition to these basic considerations regarding the repository content, we had further design rationales in mind. The most important ones can be subsumed as follows:

C5: **Start simple, allow later refinement:** The basic idea was to keep the initial structure simple, but extensible. We wanted a running repository as soon as possible to gain user feedback early in the course of the project. Based on the experiences gained while filling the initial repository, the underlying schema then should be refined and/or corrected if necessary.

As depicted in Fig. 2 to Fig. 4, the resulting internal ViSEK schema is hierarchically structured and subdivided into several levels. Starting from the top level (level 0), which contains entities for precise context descriptions of the stored knowledge (compare to C2), we refine and detail the schema with lower levels holding more specific entities[3]. Consequently, knowledge and experience are represented by a set of entities in the internal ViSEK repository structure. These entities are connected according to predefined relations.

[3] Note that it is not necessary that each and every entity of level *n* must be refined in the next lower level *n+1*.

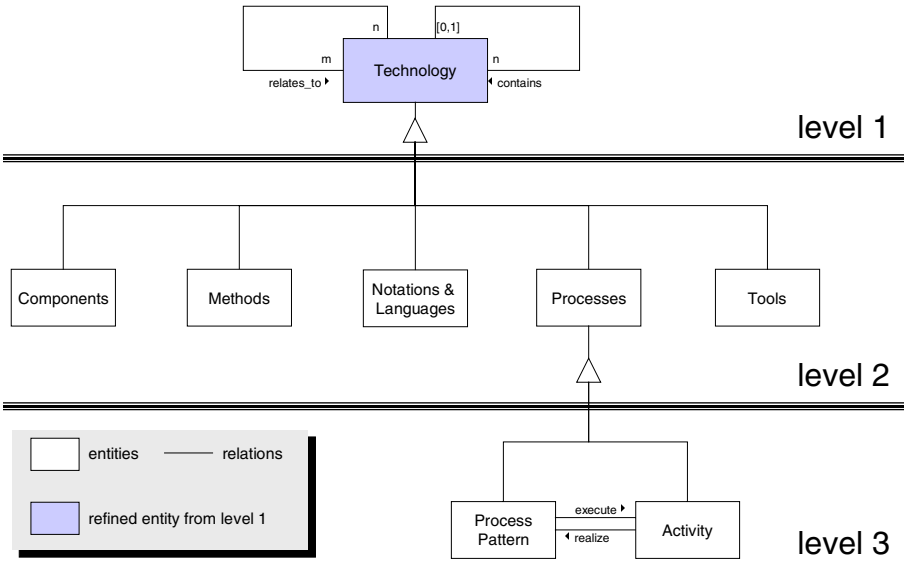

Fig. 3. ViSEK's internal repository schema: level 2 refinement of entity "Technology"

If not mentioned otherwise, relations are bi-directional to allow easy navigation in the repository. However, in Fig. 2 to Fig. 4, we only provide the name of relations in one direction to reduce complexity. For the representation of the schema we use a UML-like notation. Cardinalities are expressed by using the numbers *0* and *1,* and the letters *n* and *m*, were *n* and *m* stand for any integer number including zero. Hence, a relation described as 0:n, 1:n, or n:m may or may not exist for a stored knowledge or experience description.

Each instance of an entity is represented by a list of attribute value pairs. Attributes of an entity can be subdivided into the following distinct groups:

1. attributes describing *content* (i.e., technologies, concrete experience, or background information)
2. attributes describing *context* (e.g., application domain or intended user group)
3. attributes describing *significance* (e.g., how useful is a technology description for a concrete user)
4. attributes for *managing the repository* (e.g., user access rights, etc.)

Note that if, in some case, certain characteristics can not be represented with the initial set of entities and their attributes, the suggested schema is easily adaptable (compare to C5). This can be achieved by either locally adding attributes to a specific entity, or by introducing further entities and specific relations. Stored knowledge and experience descriptions that do not use this specific entity will not be affected all.

Top level (level 0) entities: These entities for storing descriptions of the "*Basic Context*" (e.g., the intended user group, or legal & technical restrictions), the application "*Domain*", or the "*Significance*" are inherited to all knowledge and experience entries in the ViSEK repository. The "*Problem*" entity allows the storage of problem descriptions for which a "*Knowledge Element*" (i.e., stored knowledge or experience)

may be used. In addition, the level 0 entity "*Project*" allows the storage of project information for providing usage examples (compare to C3), or the concrete project in which a lesson learned (i.e., experience) was gained (compare to C4).

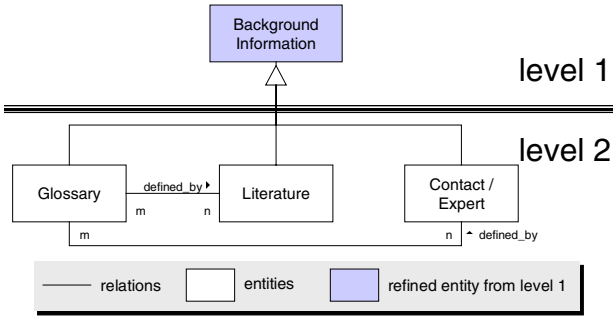

Fig. 4. ViSEK's internal repository schema: level 2 refinement of entity "Background Information"

The Level 1 entities: While entities on level 0 are used to commonly characterize all repository entries, the entities of level 1 are used to classify the stored descriptions. As illustrated in Fig. 2, all of them are directly derived from the level 0 entity "*Knowledge Element*", and therefore, inherit the precise context description. Currently, we differentiate between three different level 1 entities:

1. *"Technology*": This entity is used to store all knowledge descriptions, the main entries of the ViSEK portal (compare to C2). As illustrated in Fig. 3, it is further refined into entities for storing "*Components*" (i.e., code, or patterns), "*Methods*" (e.g., for testing), "*Notations & Languages*" (e.g., UML), "*Processes*" (e.g., How to apply UML), or "*Tools*" (e.g., Rational Rose, or the Gnu C++ compiler). For process descriptions we use the approach suggested by [5]. Therefore, the "*Processes*" entity is further refined on level 3 into an entity for storing "*Process Pattern*" and "*Activity*" descriptions.
2. "*Experience*": This entity is used to store lessons learned, the only form of experience that is stored in the ViSEK portal (compare to C4). Since experience usually is gained with or about a certain technology, a relation between the "*Experience*" and "*Technology*" entity is defined to express about which technology a certain lesson learned was gained. Information about the project in which the lesson learned was "*gained_in*" is stored with the help of the predefined relation.
3. "*Background Information*": This entity is used to store all additional information in the ViSEK portal (compare to C3). As illustrated in Fig. 4, it is further refined into entities for each type of information mentioned in C3. One could argue that since the entity only stores information, it should be part of level 0. However, we decided that the entity should be placed on level 1, and thereby, can inherit all descriptions from the level 0 entities. This allows greater flexibility in some cases and improved quality of the stored information. For instance, it is possible to precisely list the domain(s) for which an expert is listed, or to describe how useful a certain type of information is for repository users by providing an instance of the entity "*Significance*".

This closes our description of the current structure used for the ViSEK portal. Because of the lack of space, we must refer to [7] for a more detailed description, together with an extended example of how a technology description (i.e., content) is stored using the schema. However, we can state from the first practical experiences while running the repository that the structure has been accepted by our users. Some adoptions, based on the users' experiences, were necessary but could easily be integrated into the storage structure.

5 Conclusion and Future Directions

Although SMEs typically do not have the resources to invest in conventional training programs and for testing new technology, continuous long-term learning is crucial for their survival. The repository structure of Germany's ViSEK portal is targeted towards the specific needs of SMEs by delivering technical information in a goal-oriented way and relating it to practical experiences. The repository structure itself is kept lightweight and leaves flexibility for future extensions and refinements.

The ViSEK Portal was first presented at the CeBit fair in Hannover in March 2002. Within 7 days more than 100 visitors demonstrated serious interest in ViSEK and its information services by signing up for trial membership. We received positive feedback on the concept of an online SE center of competence and the way SE knowledge is going to be presented.

Undoubtedly, our major future challenge is to fill the repository with suitable content. First experiences indicate that the material at hand, i.e., technical reports and research work, is not suitable for the ViSEK repository without change. Significant efforts will be necessary to process this material for publication to SMEs as either a technology description, experience or problem. Furthermore, it seems that this work has to be done by experts of the respective fields, because non-experts seem not to be able to extract and condense the information as needed.

Besides this, we will continuously evaluate the acceptance and usefulness of the information provided by ViSEK to SMEs as we continue to fill the repository with content that is either requested by SMEs or can be expected to be of use. By this we hope to be able to gain further experience on how to effectively support long-term learning of "agile" SMEs.

Acknowledgements

Part of this work has been conducted in the context of the Sonderforschungsbereich 501 'Development of Large Systems with Generic Methods' (SFB 501) funded by the Deutsche Forschungsgemeinschaft (DFG) and as part of the ViSEK project supported by the Department of Education and Research (bmb+f) of the German Federal Government.

References

1. A. Aamodt, M. Nygard: Different roles and mutual dependencies of data, information and knowledge – an AI perspective on their integration. In: *Data and Knowledge Engineering*, 16:191–222, 1995.
2. V.R. Basili, G. Caldiera, D. Rombach: Experience Factory; In J.J. Marciniak (ed.), *Encyclopedia of Software Engineering*, vol 1, 469–476; John Wiley & Sons; 1994.
3. M. Broy, S. Hartkopf, K. Kohler, D. Rombach: Germany: Combining Software and Application Competencies. In *IEEE Software*, 18(4), July/August 2001.
4. CeBase: NSF Center for Empirically Based Software Engineering. Homepage @ http://www.cebase.org. visited May 2002.
5. M. Gnatz, F. Marschall, G. Popp, A. Rausch, W. Schwerin: *Modular Process Patterns Supporting an Evolutionary Software Develeopment Process*. In: F. Bomarius, S. Komi-Sirviö (eds.), Product Focused Software Process Improvement. 3rd International Conference, PROFES 2001, Kaiserslautern, Germany, September 2001, Lecture Notes in Computer Science (LNCS) No. 2188, Springer, 2001, 327–340.
6. R.L. Feldmann: On Developing a Repository Structure Tailored for Reuse with Improvement. In: G. Ruhe, F. Bomarius (eds.), Learning Software Organizations: Methodology and Applications, Lecture Notes in Computer Science (LNCS) No. 1756, Springer, 2000, 51–71.
7. R.L. Feldmann, I. John, M. Pizka: ViSEK Repository Schema. ViSEK report 003/E, 2002, Virtuelles Software Engineering Kompetenzzentrum. On-line @ http://visek.de
8. S. Henninger: Supporting the Construction and Evolution of Component Repositories. In: *Proc. of the Eighteenth Int. Conference on Software Engineering*, 279–288. IEEE Computer Society Press, March 1996.
9. F. Lehner: How do Companies Learn? Selected Application from the IT-Sector. Keynote given at the 3rd International LSO Workshop, University of Kaiserslautern, Germany, September 2001. (On-line @ http://wwwagse.informatik.uni-kl.de/LSO2001/Lehner.pdf).
10. M. Lindvall, M. Frey, P. Costa, R. Tesoriero: Lessons Learned about Structuring and Describing Experience for Three Experience Bases. In: K.-D. Althoff, et. al. (eds.), Advances in Learning Software Organizations, Lecture Notes in Computer Science (LNCS) No. 2176, Springer, 2001, 106–119.
11. C. Tautz: Customizing Software Engineering Experience Management Systems to Organizational Needs. PhD thesis, Department of Computer Sciences, University of Kaiserslautern, March 2000.
12. ViSEK: Virtuelles Software Engineering Kompetenzzentrum. Homepage @ http://visek.de, visited May 2002.

Tool Support for Experience-Based Methodologies

Scott Henninger

Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
scotth@cse.unl.edu

Abstract. Experience-based approaches to software development promise to capture critical knowledge from software projects that can be used as a basis for continuous improvement of software development practices. Putting these ideas into practice in the quickly evolving discipline of software engineering has proven elusive. Techniques and tools are needed that help software practitioners apply past knowledge to current projects while engaging in knowledge creation processes. This paper outlines the experience factory and organizational learning approaches, both of which explore how experience-based approaches to software development can be used to improve software development practices. A software tool is used to investigate how these two approaches can be integrated to create an approach that addresses many issues of knowledge management in the software engineering field.

1 Experience-Based Approaches for Software Engineering

An experience-based approach to software development seeks to draw on past experiences as a resource for planning and executing software development efforts [11]. By drawing on experiences, these techniques must necessarily be able to address diverse development needs [45] arising from the many application types and development tools and approaches available in the modern software developer's toolbox. The objective of an experience base is not a matter of trying to find a single universally applicable development approach and/or programming language, but one of understanding the circumstances in which a given technique, tool, methodology, etc., is most appropriate. Developing this understanding requires an empirical approach, one that carefully collects observations from experience and establishes causal relationships between contextual factors and outcomes [11].

The knowledge generated within each organization to specialize existing techniques to the point that software products can be developed represents a significant corporate asset. Knowing how the Capability Maturity Model is structured or how Enterprise JavaBeans can be used to create applications is one form of knowledge that is found in textbooks. But understanding how those tools and techniques can be used to create a payroll application for a specific industry or corporation is a more valuable kind of knowledge that is organization-specific and consequently cannot be found in textbooks. This knowledge is both created by and used within the specific context of a software development organization.

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 44–59, 2003.

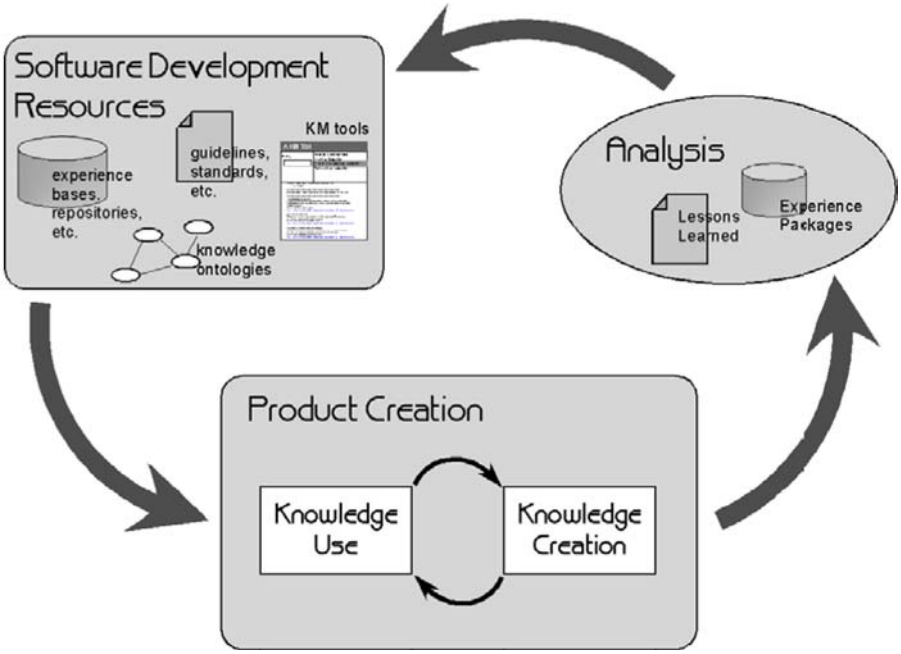

Fig. 1. The Experience-Based Knowledge Lifecycle.

While frameworks for experience-based approaches have been developed [9, 15, 23], little research has been performed on how this knowledge can be captured and systematically employed within an organizational context. Tools and techniques are needed that capture software development experiences and make them available for subsequent development efforts. This paper explores the intersection of two approaches that begin to address these issues, the Experience Factory [8] and Organizational Learning [41], and demonstrates these concepts through an exploratory prototype that supports the capture and use of project experiences.

2 Experience-Based Knowledge Management

An experience-based approach to software development involves using an organization's accumulated knowledge of the development process and application domains as the basis for planning, design, and implementation. Figure 1 depicts the experience-based knowledge lifecycle as one of using software development knowledge to create new products. Knowledge can come in many forms, including guidelines, standards, experience packages, pattern languages, manuals, software development ontologies, and other forms of knowledge. During product creation, new knowledge is created to meet development needs that are unique to the project. This new knowledge can be synthesized and packaged in an analysis phase to create new knowledge resources. The newly formed knowledge is then used as the basis for new product development efforts.

Within product creation lies another important cycle, where existing knowledge is brought to bear and extended to create new products. The assumption is that because each project is unique, past experiences can only be used as a resource, a guide that ensures improvement while avoiding known pitfalls. While some experiences, design artifacts, planning modules, etc., can be reused, each project has unique characteristics and requirements that extend the available knowledge store.

The exploration of how existing knowledge is brought to bear on a problem and used to create new knowledge [46] during product development has yet to be pursued on any kind of scale, particularly in the context of software engineering. Many information retrieval and data minim tools have been created, but not tools that integrate knowledge reuse into the process. Experience Factories and Organizational Learning address many of these issues. Integrating the ideas embedded in these approaches seems a worthwhile endeavor that is explored in the following sections.

2.1 The Experience Factory Approach

The Experience Factory is a framework that explicitly addresses validating processes and technologies in the context in which they are used [8, 11]. This approach divides software development efforts among organizations with separate responsibilities of developing projects and capturing experience. The Experience Factory unit is responsible for developing, updating and providing reusable experiences that can be utilized by product development teams. Experience artifacts can be generated either through requests by the product development units or through independent analysis of existing projects [10].

The main focus of the Experience Factory, to "make reuse easy and effective" [10], is accomplished through an infrastructure to produce, store, and reuse experience. It uses the Quality Improvement Paradigm (QIP) that consists of six steps and two feedback cycles [8]. The steps, performed within projects, are 1) characterize project goals, 2) set goals, 3) choose a process, 4) execute, 5) analyze results, and 6) package the results of analysis. The two feedback cycles are a project feedback cycle (feedback provided during the execution phase of the project) and an organizational feedback cycle (feedback provided to the organization during and at the completion of the project). The definition of the QIP implies that organizational and product improvement requires continually accumulating knowledge, and storing that knowledge in a form that can be easily accessed, used, and modified.

Note the close relationship between Figure 1 and the QIP. The software development resources shown in Figure 1, captured from previous development efforts, are used to characterize and set project goals. Processes are chosen and executed during project creation. During execution, the results of the development effort are analyzed and packaged as new resources that can be utilized by subsequent development efforts. The two QIP feedback loops are represented by the outside cycle at the organizational level and the project level feedback in the product creation phase.

The Experience Factory is designed to support the entire process of knowledge management, including accumulating experience from the project organization, evaluating those experiences, repairing faults, and extracting items that need to be stored. But current tools to support the Experience Factory have focused almost exclusively on repository technology for experience packages [3, 9, 11, 50]. Although it is recognized that a high level of intuition is required for the analysis phase,

making it extremely difficult to automate [4], analysis tools have received some attention, particularly in terms of lessons learned repositories [2, 4]. Some work has also been done on tools for measuring the effectiveness of repositories [28] and using user feedback to calculate the perceived usefulness of knowledge assets [5]. But little work to date has concentrated on tools to support the (re)use of experiences, and develop methodologies that explicitly integrate prior experiences and lessons learned into the development process.

2.2 The Organizational Learning Approach

The organizational learning approach to software development [41] captures project-related information during the creation of individual software products. It is designed to disseminate knowledge as it is created in the organization so people can begin to build a culture based on success, avoid duplicating efforts, and avoid repeating mistakes [39]. Subsequent projects can then use this information by searching for issues and problems in a repository of project experiences. The repository contains significant problems addressed in a project along with the solution and method used by the project. Projects with similar characteristics can be retrieved and assessed for applicability to the problem at hand. Information in the repository can not only point to reusable solutions and code, but also suggest how novel problems can be approached by adapting similar problems, warn of possible problems with solution methods, and help designers understand the boundaries of a problem. These principles have been demonstrated through an exploratory prototype, named BORE (Building an Organizational Repository of Experiences) [36].

The approach also includes a continuous improvement process where knowledge is refined as projects use and extend the repository [35]. But support for this process in BORE was particularly weak. Pilot projects using the tool revealed that people needed more guidance on what information should be captured and when the repository should be consulted. In other words, having a repository and a search engine provided inadequate support for the overall approach. More recent activities, reported here, have coupled software methodology support with the repository. This allows knowledge to be delivered to the development team through the methodology; avoiding pitfalls of people being unable to fully articulate their query [13] or know when knowledge exists that can aid problem-solving methods. In addition, adopting an experience factory approach guides repository evolution to ensure high quality, broadly applicable, processes are used and refined through use.

3 Experience-Based Repositories

Knowledge repositories in fast evolving domains, such as software development, face becoming obsolete almost as quickly as information is captured. Experience-based approaches “try to avoid this by enriching the best practice description with experience gained in its application” [50]. To accomplish this requires repository technology capable of representing both knowledge categories and associated experiences.

Relational models have been used to represent attributes of specific packages [9], and object-oriented models have been developed [51]. These representations allow the definition of types and attribute-value pairs to describe the types. Case-based

technology further builds on attribute-value representations by applying similarity matching techniques and a representation-adaptation-capture cycle addressing continuous knowledge acquisition [40].

The fact that knowledge in software development settings is both dynamic and situation-specific [23, 36, 40, 57] indicates that a case-based approach is a good match in this setting [36, 57]. A case is a "contextualized piece of knowledge representing an experience" [43], making case-based decision support an excellent choice to represent experience-based software engineering methodologies, tips, and techniques. Case-based decision support technology [43, 54] for capturing project experience-based repositories also provides the basis for understanding the context in which specific methods will work.

The case-based reasoning cycle [1] naturally incorporates building on past experiences in a continuous learning process. The CBR cycle first retrieves from a case base, which ensures knowledge use as shown in Figure 1. The chosen item(s) are then revised or otherwise adapted to the current problem setting. Then the newly adapted case is stored to accumulate the newly gained experiences.

A remaining issue in repository and case-based techniques is the passive nature of how they are utilized in the development process. Searching is seen as an extra and optional step that is abstracted from the normal workflow and used on a discretionary basis. People do not always know that information exists that may be of help to them and often have trouble articulating the information need in terms the system understands [13, 24, 30]. Simply having a repository of experiences and/or lessons learned is insufficient to get knowledge to the people that may benefit from it. Mechanisms are needed that trigger or otherwise alert people to knowledge that is available and relevant to the task at hand.

4 The Bore Experience Base Tool

BORE (Building an Organizational Repository of Experiences) is a prototype tool designed to further explore and refine the requirements for tools supporting experience-based approaches. Its purpose has been as a proof-of-concept that is used to articulate organizational learning and experience-based software development techniques. The tool has evolved from exclusively focusing on repository technology [38] to using defined software methodologies as the organizing principle for the technology [39]. It combines a work breakdown structure with repository tools to create a framework for designing software process methodologies and repository technology for capturing and applying knowledge artifacts. The BORE tool and methodology provide additional support for Experience Factory concept through applicability rules for activities, support for process modeling and enactment, and case-based organizational learning facilities.

The BORE prototype is a Web-based applet using a three-tiered architecture with Java AWT for the interface, Java for the client application logic and server, and an SQL database back-end. BORE has two main interfaces, shown in Figure 2. The Case Manager, shown to the left in Figure 2, displays a hierarchical arrangement of cases that define the activities in a project's development process. In the figure, a project named "ProjectX Demo" has been chosen from the list of resources in the drop-down menu that displays projects. Each of the nodes in the project hierarchy,

which are cases corresponding to project activities, can be opened to view information about the activity. In the window to the right in Figure 2, the activity named "0.10.0 Choose Project Methodology" has been opened.

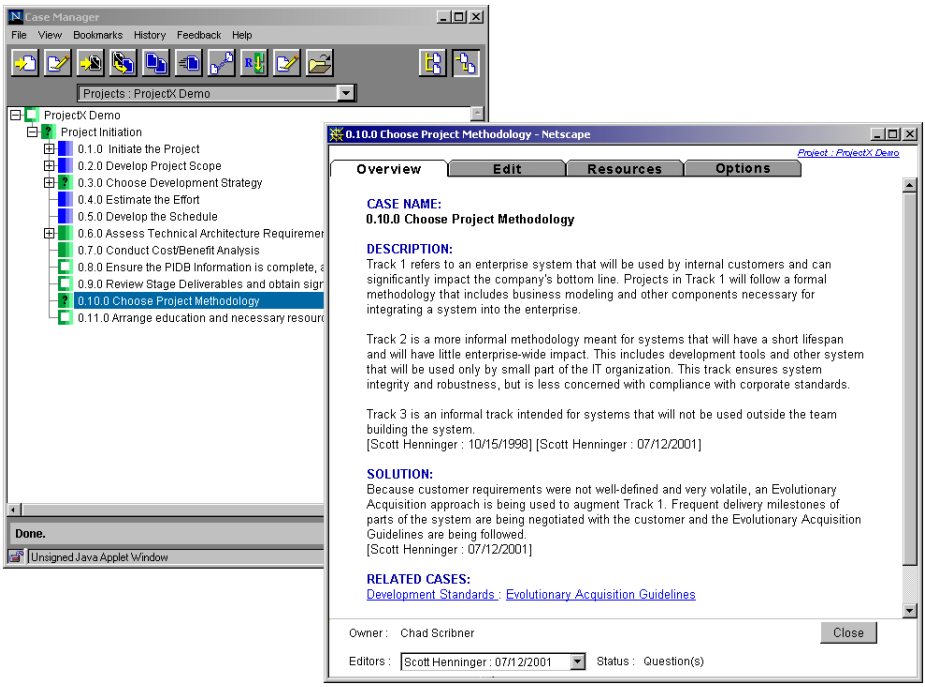

Fig. 2. The BORE Case Manager (left) and a Case Window(right).

The case-based architecture of BORE uses cases to represent all primary information sources. Cases consist of a problem description, solution statement, and some ancillary information, such as links to related cases. The primary use of cases is to represent project activities, such as the one shown in Figure 2. The solution field of a case is used to document project-specific information accumulated while executing the activity. The Edit tab is used to edit the activity fields and the Resources tab is used to attach documents to the activity. The Options tab, which will be described in detail later, is used to choose process options available to the activity. Activity status is kept for each activity and is displayed as a color-coded icon in the Case Manager (active, resolved, etc.), thus providing a way to get a visual overview of a project's status.

These core features represent a case-based organizational memory paradigm of BORE [36]. Each case represents a project-specific activity that is used to help coordinate and disseminate project experiences. The project hierarchy can be used as a dynamic checklist of project activities that capture information about projects in an organization-wide repository. This creates a repository of experiences that software development efforts can utilize to find potential solutions to development problems.

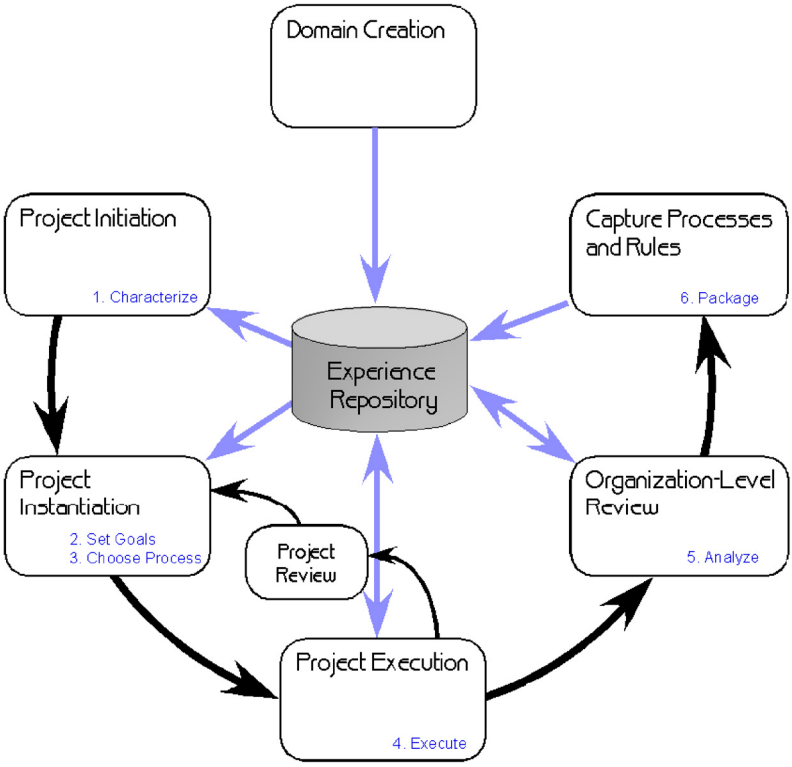

Fig. 3. The BORE Experience-Based Methodology.

4.1 The BORE Methodology

Using BORE to manage software development processes and methodologies involves the experience-based methodology shown in Figure 1. The figure shows relationships between the steps in the methodology and, where applicable, the six QIP steps described earlier in this paper are shown in the bottom-right of the activity bubbles.

To use BORE, the organization first creates a domain by creating a work breakdown structure of activities and rules that define when an activity should be assigned to a given project. Once defined, this domain will be used to create project instances and will be modified through changes to the experience base. The next step is to create an instance of the domain for each project using the system. This creates a set of project initiation activities as defined in the domain. The third step is for project personnel to begin executing the activities and document their progress in the activities assigned to the project. BORE allows activities to be further broken down by associating tailoring options to any of the domain activities. Choosing options may assign other work breakdown structures to the project, thereby refining the activities that the project will undertake.

The result is a tailored development process consisting of activities to be executed by the project team. Once instantiated, the activities are used to document project

specific issues. Project documents and code artifacts, templates for which can be defined in the domain and copied to projects, are attached to project activities to act as a repository for document management. The process of choosing tailoring options, executing the assigned activities, and periodically reviewing the project, defines the normal project development cycle, or spiral [18].

While executing, project members are creating and modifying project artifacts and further tailoring the development process. This may entail deviating from the process as defined in the domain. A critical difference between this approach and others[21] is that deviations are knowledge creation points that can be analyzed and potentially added to the repository. Deviations are seen as a normal part of the development process, one that leads to refinements in the experience base, organizational processes, etc. The activities are then packaged and placed in the repository with appropriate rules that determine what kinds of projects should use the newly defined activities. The domain is thus modified through the modified experience base, completing the experience-based knowledge lifecycle. Each step of this process is explained in more detail in the following sections.

4.2 Creating BORE Knowledge Domains

Each domain in BORE is a model of development activities defined by a set of domain *activities* in a hierarchical work breakdown structure and domain *rules* that define when an activity should be used. A domain in BORE can be as simple as a repository of cases related to a topic or as complex as a development methodology for a software development organization. Projects belong to a single domain, which is chosen when a project is created in BORE. All subsequent project activities will use the cases and rules defined for that domain. Allowing multiple domains supports scalability and allows organizations to partition development activities into domains that mirror diverse environments or product lines.

Domain activities define the space of possible activities for projects within a given domain. Activities can be as high-level or detailed as desired. The Case Manager in Figure 2 shows a set of activities for project initiation, taken from the development process of a medium-sized IT division for a large corporation.

The defined process can be tailored to meet project-specific needs through domain rules, which define a set of conditions under which a project is required to execute a specific activity. In other words, the domain rules choose which domain activities a project must follow to conform to the process standard. The standard may therefore define a broad set of activities and the domain rules determine which of those activities are applicable to a given project. Rules are defined through a set of preconditions, defined by question/answer pairs, and actions, that are fired when all preconditions evaluate to true.

4.3 Project Instantiation

When a project is created in BORE, an instance of the process is created. This involves creating a set of project initiation activities that are specified in the domain's initialization rule and copying the associated domain activities into the project hierarchy.

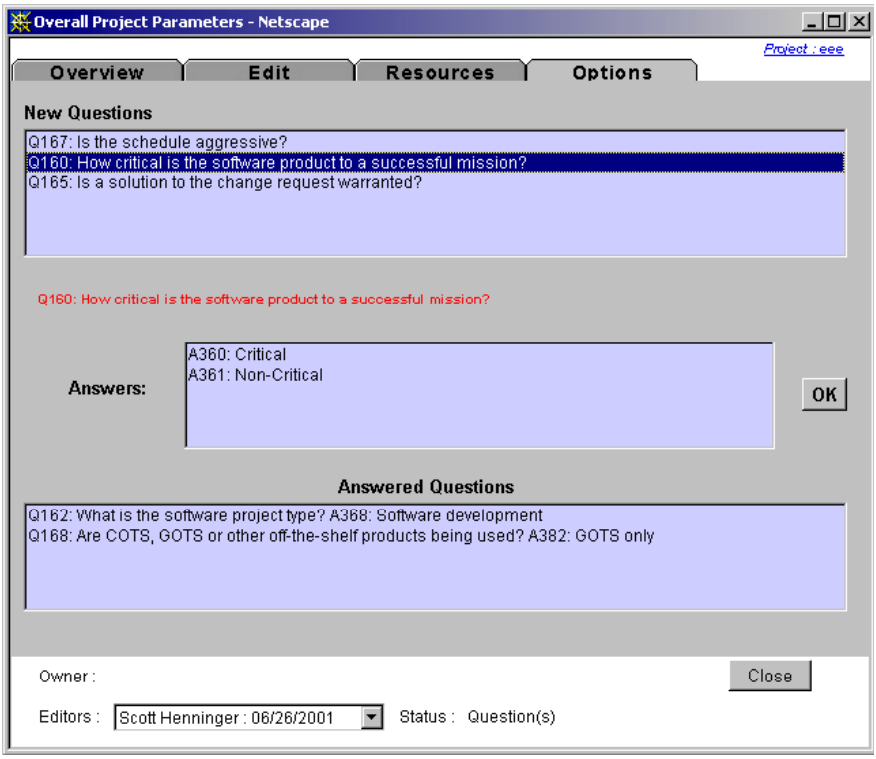

Fig. 4. The Options Tab.

The initial activities represent a first pass to defining project activities than can be further broken down through options that can be attached to any of the domain activities. Activities that have options associated with them are denoted by the "?" in the icon next to the activity label in the Case Manager (see the project hierarchy in Figure 2). The options are found by opening the Options tab (Figure 4) and are used to tailor the process to the project's unique characteristics, usually by breaking down the activity into constituent sets of subtasks.

For example, the development process in the NASA Goddard matrix-based methodology [55] defines a number of key tailoring criteria as shown in Figure 4, such as whether the project is "critical" or "non-critical." The initiation process in this methodology creates an activity named Overall Project Parameters that holds options that, in this example, will determine what kinds of design reviews will be held by the project. The same technique could be used to support other tailorable methodologies such as Cockburn's Methodology Grid [19].

In Figure 4, two questions have been answered and three more remain in the New Questions stack. When a question is chosen, the possible answers are displayed in the Answers field in the middle of the window. Selecting the answer moves the question/answer pair to the Answered Questions field. Answering a question can potentially fire a rule, adding activities to the project or adding/subtracting questions to/from the New Questions queue. Answers can be changed or un-answered, allowing users to easily engage in a what-if dialogue with BORE. Choosing a different

answer causes BORE to backtrack the actions previously executed and executing the actions required by any rules fired by the new question/answer pair.

A key feature of this tailoring process is the flexibility offered by allowing options to be associated with any domain activity. This allows processes to be defined that iteratively expand each part of the project's process as they gain the knowledge necessary to address process issues [52]. Initial projects can define activities with options that add new activities that in turn have options to further refine the process and provide resources for the project. In this way, the project tailors the overall process to its needs when it is able to address the issues involved. In addition, varying levels of detail can be supported by defining options as deeply in the tree of domain activities as desired.

4.4 Project Execution

During project enactment, project personnel perform the work specified in the activities, documenting work by either editing the activity (in the solution field, shown in the case window of Figure 2) or attaching work products to the activity. Although many formalisms have been created for software process automation [7, 31, 56], this has not been the focus of the BORE work. Instead, we have created rudimentary tools to support basic functions, such as version and source change control, and have instead focused on the means to represent contextual factors for applying defined processes and activities.

One way to document project-specific activities is to edit the Solution field of the Case Window (see Figure 2). This is intended as a quick and easy way to annotate an activity, although any amount of documentation is allowed. A second method of project documentation is found in the Resources tab (see the Case Window in Figure 2), which allows documents of any type to be attached to an activity. Using the browser's HTTP facilities, the document can then be downloaded and opened with a simple mouse click. Document modifications are supported through a version management system integrated into BORE.

BORE also supports assigning project personnel to activities. When domain rules are created, roles can be assigned to activities when they are instantiated in a project, as shown in the Case window of Figure 2. Each domain defines a set of roles, and projects can assign one or more persons to the role. Activities can thus be assigned to specific project personnel for execution.

4.5 Analysis and Review Processes

Analysis and review of process activities can occur at two distinct levels. The first is at the project level, where it is anticipated that each project has unique characteristics and will therefore need to deviate from the process in a significant way. Note that the assumption here is that more detailed process definitions are possible with this technology than with traditional, high-level, process definitions. Instead of simply stating that a requirements definition document is required, this approach allows organizations to define best practices for eliciting and defining requirements, potentially for different kinds of applications. Knowledge at this level of detail has potential for supporting software developers and managers in their daily activities. Note that this approach is a combination of knowledge manageenvisionedment and methodology support.

Process deviations are supported by allowing projects to define new activities for their project. In a disciplined environment, such deviations would go through a project-level review, as shown in the project instantiation and review cycle of Figure 1. The outcome of such a review can have one of a number of implications. Appropriateness of chosen options can be discussed, perhaps causing one or more of them to be changed. Another is that the deviations are deemed inappropriate and the project is directed toward accepted development practices of which project personnel may have been unaware.

The most significant outcome of a review, one that all projects will encounter, is that the domain needs to be extended to meet emerging software development needs. For example, suppose a project is the first in the organization to use Java applets, and designs a method to use JDBC to access Oracle tables. To do this means deviating from the process because certain issues of creating a Web-based database server haven't been addressed before in this organization. The project will supply deviation rationale (in this case that JDBC for Oracle tables is needed) and create activities to perform the necessary tasks.

Individual projects can create new activities, but escalating these activities to the domain should be performed by a person or team with organization-wide responsibility for the process. Thus, a second level of analysis and review is necessary to ensure that chosen processes have shown utility and can be successfully used in other projects.

4.6 Experience Packaging

Packaging project experience is a two-step process of creating the work breakdown structure of activities and defining rules for when activities should be applied to a project, bringing our process back to the domain definition phase in Figure 1[1]. In our example, new domain activities, such as "Create Oracle configuration files" or "Evaluate JDBC COTS software," would be created and added to the domain in a task/subtask breakdown. The deviation rationale, in this example stating that the process needs to have activities for projects using JDBC and Oracle with a Java applet, is translated into rules. New questions are created and associated with the proper domain activities. For example, a domain activity on the software architecture may add an entry for Java applets and a question on database back-ends. Choosing these options will assign the newly created activities on using JDBC to the project. In this manner, the project's experiences are used to pave a new path that other projects can follow as part of the refined process.

5 Open Issues and Future Work

Early prototyping and work with software development organizations has demonstrated that the BORE approach is capable of capturing existing methodologies and software development knowledge, and provides possibilities, such as capturing

[1] Note that we have purposefully omitted an explicit link from packaging to domain creation in Figure 3. We wanted to show that projects refine the domain through the experience base. Each project therefore draws on the collective definition of the process by reusing experience-based knowledge placed in the repository and structured through the domain model.

knowledge through the development process, that organizations had not previously envisioned [39]. But there is an inherent struggle between understanding the inherent complexity of diverse development contexts and the desire to create a standard that is easily understood by managers and developers alike. The BORE tool provides some relief for this tradeoff. Project personnel do not need to understand the methodology in its entirety, just those elements the project is responsible for. But the burden of understanding a more complex process remains for personnel with methodology maintenance responsibilities. Better tools are needed to supply higher-level views of activity structures and relationships between rules.

Maintaining the BORE repository requires personnel that specialize in software process and maintaining rule bases. Many organizations, fueled by the need to conform to the CMM or other standards, are already creating Software Engineering Process Groups (SEPGs) or otherwise dedicating some effort to maintaining process standards. BORE can serve as a tool to assist these groups in the creation and maintenance of the organization's process. Rule-based expertise is also needed to maintain BORE's rule base. Complex rule structures can be as difficult to maintain as software systems, and require similar techniques, such as version control and test plans. Organizations often find it difficult to justify the overhead of such positions and continued effort is needed to minimize documentation overhead while further understanding the benefits gained by investing in organizational learning and experience factory strategies.

The BORE system currently relies on rules to find relevant activities for projects. To the extent that the activities can be designed properly, this will work. But there may be instances where a search for related activities may be useful. BORE has a simple search tool, but we are investigating more sophisticated case-based retrieval. Hierarchical case-based planners [49], which generate plans based on case-based search criteria and options, seem particularly well suited to the kinds of work we are pursuing. We hope to draw on some work that integrates rule-based and case-based systems [33, 48] to find work breakdowns that are relevant to the characteristics of a project.

Integrating languages and formal process models into environments has been researched in a number of contexts [6]. These process-based environments, often called Process Centered Environments (PCEs) often cover both the process modeling and process enactment phases of the process engineering lifecycle [31, 34]. Some of these environments have investigated different formalisms, such as knowledge-based [14] and Petri net [7] approaches, while others have focused on elements of software development, such as hierarchical decomposition of activities [12, 29], collaboration and coordination [25] [32], and analysis and simulation [47]. These techniques contain process representations for creating processes and automatable process actions [20, 22, 42, 44]. The BORE system has focused more on knowledge management and supporting experience factory approaches. Nonetheless, the use of a process language would enable the integration of many automation features and future work will look into this, perhaps by embedding an ontology-based language in the BORE rule engine.

Some of the research questions we will continue to explore with BORE include designing work practices centered around case-based organizational learning processes. Design tools must reward users using the tool with high-quality knowledge, as well as leaving information for subsequent development and maintenance efforts. We believe these are compatible goals. Long-term projects need to track their progress and deci-

sion making process, but the challenge is to capture this knowledge in a form that can be used in other development efforts. Some major research questions include: designing methodologies based on standards set by previous projects yet flexible enough to accommodate changes in business needs and technology; creating a development process that begins by using the resources supplied by the repository and updates the repository as the project progresses; finding the "right" level of documentation [16, 53] in techniques such as agile methodologies that do not overwhelm developers yet leaves a trace for subsequent efforts; and organizational changes needed to make such techniques work in the organization (i.e. the technology transfer and tool adoption problems).

Most important is the extremely difficult task of getting organizations to take part in efforts to evaluate emerging research in realistic settings. The BORE system is beginning to move out of the prototyping phase and is currently being evaluated by a local IT department, as an implementation technology for the MBASE methodology at the University of Southern California [17], and the Software Design Studio at the University of Nebraska-Lincoln [26]. In addition, other development organizations, including NASA Goddard [37] have shown interest and may engage in further pilot studies that will be the subject of subsequent research papers.

6 Conclusions

The research described here investigates the integration of experience factory and organizational learning techniques. The techniques complement each other, with the experience factory supplying the organizational structure for use, capture, and packaging of knowledge, and the organizational learning approach providing case-based tool support and a mechanism for explicitly defining software development methodologies. The overall goal is to develop a tool and methodology that is based on the use of best practices embodied in a defined software development process, but that allows projects to improvise when necessary. The term "improvise" is key here, as it connotes the fact that innovation occurs within a known framework [27] – the process repository of best practices, in our case.

To achieve these goals, we have coupled process and technology to turn defined development processes into dynamic "living" resources [58] that can be extended and improved as new project needs and application requirements emerge. General principles are captured in the methodology and cases capture situation-specific knowledge of actual practice. The emerging, case-based, knowledge can then be turned into standards representing a software development organization's best practices that can be brought to bear on individual development efforts. As the repository accumulates through principled evolution of the methodology, it improves to able to handle a wider range of circumstances [43], while evolving toward answers to problems that fit the organization's technical and business context.

Acknowledgments

I gratefully acknowledge the efforts a number of graduate students that have helped develop BORE, including Kurt Baumgarten, Kalpana Gujja, Rishi Kumar, Yu Li, Sarathkumar Polireddy, and others. This research was funded by the National Science Foundation (CCR-9502461, CCR-9988540, and ITR/SEL-0085788).

References

1. A. Aamodt and E. Plaza, "Case-Based Reasning: Foundational Issues, methodological Variations, and System Approaches," *AI Communications*, 7(1), pp. 39-52, 1994.
2. D. W. Aha and R. Weber, "Intelligent Lessons Learned Systems: Papers from the 2000 Workshop." Menlo Park, CA: AAAI Press, 2000.
3. K.-D. Althoff, A. Birk, S. Hartkopf, and W. Müller, "Managing Software Engineering Experience for Comprehensive Reuse," *Proc. 11th International Conference on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany, pp. 10-19, 1999.
4. K.-D. Althoff, A. Birk, and C. Tautz, "The Experience Factory Approach: Realizing Learning from Experience in Software Development Organizations," *Tenth German Workshop on Machine Learning*, University of Karlsruhe, 1997.
5. K.-D. Althoff, M. Nick, and C. Tautz, "Improving Organizational Memories through User Feedback," *2nd International Workshop on Learning Software Organizations (LSO 2000)*, Oulu, Finland, pp. 27-44, 2000.
6. V. Ambriola, R. Conradi, and A. Fuggetta, "Assessing Process-Centered Software Engineering Environments," *ACM Transactions of Software Engineering and Methodology*, 6(3), pp. 283-328, 1997.
7. S. Bandinelli, E. DiNitto, and A. Fuggetta, "Supporting Cooperation in the SPADE-1 Environment," *Transactions on Software Engineering*, 22(12), pp. 841-865, 1996.
8. V. Basili, G. Caldiera, and D. Rombach, "Experience Factory," in *Encyclopedia of Software Engineering*: Wiley & Sons, 1994, pp. 469-476.
9. V. Basili, M. Lindvall, and P. Costa, "Implementing the Experience Factory Concepts as a Set of Experience Bases," *International Conference on Software Engineering and Knowledge Engineering (SEKE '01)*, Buenos Aires, Argentina, 2001.
10. V. R. Basili, G. Caldiera, and G. Cantone, "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, 1(1), pp. 53-80, 1992.
11. V. R. Basili and H. D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments," *IEEE Transactions on Software Engineering*, 14(6), pp. 758-773, 1988.
12. N. Belkhatir, J. Estublier, and W. L. Melo, "Software Process Model and Work Space Control in the Adele System," *2nd International Conference on the Software Process*, Berlin, FRG, pp. 2-11, 1993.
13. N. Belkin, "Helping People Find What They Don't Know," *Comm. of the ACM*, 43(8), pp. 58-61, 2000.
14. Z. Ben-Shaul and G. E. Kaiser, "A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment," *Proc. Sixteenth International Conference on Software Engineering*, Sorrento, Italy, pp. 179-188, 1994.
15. A. Birk and F. Kröschel, "A Knowledge Management Lifecycle for Experience Packages on Software Engineering Technologies," *1st International Workshop on Learning Software Organizations (LSO 1999)*, Kaiserlautern, FRG, pp. 115-126, 1999.
16. B. Boehm, "Get Ready for Agile Methods, With Care," *Computer*, 35(1), pp. 64-69, 2002.
17. B. Boehm, B. A.W., D. Dincel, and D. Port, "Guidelines for Model-Based (System) Architecting and Software Engineering (MBASE) v 2.2," Univ. of Southern California January 2001.
18. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, 21(5), pp. 61-72, 1988.
19. A. Cockburn, "Selecting A Project's Methodology," *IEEE Software*, 14(4), pp. 64-71, 2000.

20. R. Conradi and C. Liu, "Process Modeling Languages: One or Many?," *5th European Workshop on Software Process Technology (EWSPT '95)*, Noordwijkerhout, The Netherlands, pp. 98-118, 1995.
21. G. Cugola, "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models," *IEEE Transactions on Software Engineering*, 24(11), pp. 982-1000, 1998.
22. B. Curtis, M. I. Kellner, and J. Over, "Process Modeling," *Communications of the ACM*, 35(9), pp. 75-90, 1992.
23. B. Decker, K. D. Althoff, and M. Nick, "Integrating Business Process and Lessons Learned with an Experience Factory," *1st German Conference on Professional Knowledge Management*, 2001.
24. P. Devanbu, R. J. Brachman, P. G. Selfridge, and B. W. Ballard, "LaSSIE: A Knowledge-Based Software Information System," *Communications of the ACM*, 34(5), pp. 34-49, 1991.
25. W. Dieters and V. Gruhn, "Managing Software Processes in the Environment MELMAC," *4th ACM SIGSOFT Symposium on Software Development Environments*, Irvine, CA, pp. 193-205, 1990.
26. S. Dunbar, S. Goddard, S. Henninger, and S. Elbaum, "Bootstrapping the Software Design Studio," *Fifth Annual National Collegiate Inventors and Innovators Alliance Naitonal Conference*, Washington, DC, pp. 180-188, 2001.
27. T. Dybå, "Improvisation in Small Software Organizations," *IEEE Software*, 17(5), pp. 82-87, 2000.
28. R. Feldmann, M. Nick, and M. Frey, "Towards Industrial-Strength Measurement Programs for Reuse and Experience Repository Systems," *2nd International Workshop on Learning Software Organizations (LSO 2000)*, Oulu, Finland, pp. 7-18, 2000.
29. C. Fernström, "PROCESS WEAVER: Adding Process Support to UNIX," *2nd International Conference on the Software Process: Continuous Software Process Improvement*, Berlin, FRG, pp. 12-26, 1993.
30. G. Fischer, A. Lemke, and T. Schwab, "Knowledge-Based Help Systems," *Proc. Human Factors in Computing Systems (CHI '85)*, pp. 161-167, 1985.
31. P. K. Garg, P. Mi, T. Pham, W. Scacchi, and G. Thunquest, "The SMART Aproach for Software Process Engineering," *International Conference on Software Engineering (ICSE 94)*, Sorrento, Italy, pp. 341-350, 1994.
32. P. K. Garg, T. Pham, B. Beach, A. Deshpande, A. Ishizaki, W. Wentzel, and W. Fong, "Matisse: A Knowledge-based Team Programming Environment," *International Journal of Software Engineering and Knowledge Engineering*, 4(1), pp. 17-59, 1994.
33. A. Golding and P. S. Rosenbloom, "Improving Rule-Based Systems Through Case-Based Reasoning," *Proc. 9th National Conference on Artificial Intelligence*, Anaheim, CA, pp. 22-27, 1991.
34. J. C. Grundy and J. G. Hosking, "Serendipity: Integrated Environment Support for Process Modelling, Enactment and Work Coordination," *Automated Software Engineering: An International Journal*, 5(1), pp. 27-60, 1998.
35. S. Henninger, "Building an Organization-Specific Infrastructure to Support CASE Tools," *Journal of Automated Software Engineering*, 3(3/4), pp. 239-259, 1996.
36. S. Henninger, "Case-Based Knowledge Management Tools for Software Development," *Journal of Automated Software Engineering*, 4(3), pp. 319-340, 1997.
37. S. Henninger, "Software Process as a Means to Support Learning Software Organizations," *Twenty-fifth Annual NASA Software Engineering Workshop*, Greenbelt, MD, 2000.
38. S. Henninger, "Supporting Software Development with Organizational Memory Tools," *International Journal of Applied Software Technology*, 2(1), pp. 61-84, 1996.

39. S. Henninger, "Tools Supporting the Creation and Evolution of Software Development Knowledge," *Proceedings of the Automated Software Engineering Conference*, Lake Tahoe, NV, pp. 46-53, 1997.
40. S. Henninger and K. Baumgarten, "A Case-Based Approach to Tailoring Software Processes," *International Conference on Case-Based Reasoning (ICCBR 01)*, Vancouver, B.C., pp. 249-262, 2001.
41. S. Henninger, K. Lappala, and A. Raghavendran, "An Organizational Learning Approach to Domain Analysis," *17th International Conference on Software Engineering*, Seattle, WA, pp. 95-104, 1995.
42. G. E. Kaiser, S. S. Popovich, and I. Z. Ben-Shaul, "A Bi-Level Language for Software Process Modeling," *Proc. Fifteenth International Conference on Software Engineering*, Baltimore, Maryland, 1993.
43. J. L. Kolodner, "Improving Human Decision Making through Case-Based Decision Aiding," *AI Magazine*, 12(1), pp. 52-68, 1991.
44. H. Lee and L. J. Osterweil, "HI-PLAN and Little-JIL: a Study of Contrast Between Two Process Languages," *International Conference on Software Theory and Practice (ICS2000)*, Beijing, PRC, August 21-25, 2000.
45. M. Lindvall and I. Rus, "Process Diversity in Software Development," *IEEE Software*, 17(4), pp. 14-18, 2000.
46. J. G. March, "Exploration and Exploitation in Organizational Learning," *Organizational Science*, 2(1), pp. 71-87, 1991.
47. P. Mi and W. Scacchi, "Modeling Articulation Work in Software Engineering Processes," *1st International Conference on the Software Process*, Redondo Beach, CA, pp. 188-201, 1991.
48. H. Muñoz-Avila, D. W. Aha, L. A. Breslow, D. S. Nau, and R. Weber, "Integrating Conversational Case Retrieval with Generative Planning," *Proc. 5th European Workshop on Case Based Reasoning*, Trento, Italy, pp. 322-334, 2000.
49. D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, "SHOP: Simple Hierarchical Ordered Planner," *Proc. 16th International Conference on Case-Based Reasoning*, Stockholm, pp. 968-973, 1999.
50. M. Nick, K.-D. Althoff, and C. Tautz, "Systematic Maintenance of Corporate Experience Factories," *Computational Intelligence*, 17(2), pp. 364-386, 2001.
51. M. Oivo and V. R. Basili, "Representing Software Engineering Models: The TAME Goal Oriented Approach," *IEEE Transactions on Software Engineering*, 18(10), pp. 886-898, 1992.
52. L. Osterweil, "Software Processes are Software Too," *Ninth International Conference on Software Engineering*, Monterey, CA, pp. 2-13, 1987.
53. L. D. Paulson, "Adapting Methodologies for Doing Software Right," *IT Pro*, July/August, pp. 13-15, 2001.
54. M. Pearce, A. K. Goel, J. L. Kolodner, C. Zimring, L. Sentosa, and R. Billington, "Case-Based Design Support: A Case Study in Architectural Design," *IEEE Expert*, 7(5), pp. 14-20, 1992.
55. D. Schultz, J. Bachman, L. Landis, M. Stark, G. Meyers, S. Godfrey, and M. Tilley, "A Matrix Approach to Software Process Definition," *25th Annual Software Engineering Workshop*, 2000.
56. S. M. Sutton, D. Heimbinger, and L. J. Osterweil, "APPL/A: A Language for Software Process Programming," *Transactions on Software Engineering and Methodology*, 4(3), pp. 21-286, 1995.
57. C. Tautz and K.-D. Althoff, "Using Case-Based Reasoning for Reusing Software Knowledge," *Proc. 2nd International Conference on Case-Based Reasoning (ICCBR'97)*, pp. 156-165, 1997.
58. L. G. Terveen, P. G. Selfridge, and M. D. Long, "Living Design Memory' - Framework, Implementation, Lessons Learned," *Human-Computer Interaction*, 10(1), pp. 1-37, 1995.

Knowledge Management Support for Distributed Agile Software Processes

Harald Holz[1] and Frank Maurer[2]

[1] University of Kaiserslautern, Department of Computer Science,
D-67653 Kaiserslautern, Germany
holz@informatik.uni-kl.de
[2] University of Calgary, Department of Computer Science,
Calgary, Alberta, Canada, T2N 1N4
maurer@cpsc.ucalgary.ca

Abstract. Agile Software Development has put a new focus on the question of how to share knowledge among members of software development teams. In contrast to heavy-weight, document-centric approaches, agile approaches rely on face-to-face communication for knowledge transfer. Pure face-to-face communication is not feasible when applying agile processes in a virtual team setting. In this paper, we argue that the right approach for virtual software development teams using agile methods lies between a radical "none but source code" standpoint, and the multitude of documents proposed by heavy-weight development standards. This paper introduces work on developing a system for the task-based capture and pro-active distribution of recurrent information needs that typically arise for developers, as well as potential ways to satisfy these information needs. Our approach facilitates an incremental organizational learning process to capture and maintain knowledge on what documentation/information is actually needed, such that documentation is created on an "as needed" basis.

Keywords: task-oriented knowledge management support, virtual agile teams

1 Introduction

At first sight, Agile Software Development and Knowledge Management (KM) do not seem to fit well together. As pointed out by Cockburn [5], one of the main characteristics of agile methodologies is their attempt to shift the company's organizational and project memory from external to tacit knowledge, i.e. written documentation is replaced by informal communication among team members. While this might relieve team members from time-consuming documentation activities that are not directly relevant to their current development tasks, the absence of explicit documentation leads to a number of problems:

- Subject matter experts in larger teams find themselves spending much time in repeatedly answering the same questions.
- Team members find themselves in situations where they know that they have had a certain problem before, but cannot remember its solution.

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 60–80, 2003.

- There is no direct knowledge exchange between members of different teams if they do not belong to the same community.
- Important knowledge is lost as soon as experienced developers leave the project or company.

Although the last point is partially mitigated by the strong focus on pair programming and shared code ownership as advocated by Extreme Programming [4] and other agile methods, the other issues still remain. Thus, the advantages of following agile, light-weight methodologies have to be balanced against the disadvantages of the absence of documentation and the lack of knowledge management.

Knowledge sharing is particularly difficult in case of virtual agile teams where team members are not co-located and have less or no opportunity for face-to-face communication. On the other hand, virtual teams often use information technology, e.g. e-mail, newsgroups, on-line chat rooms or Wiki webs, to exchange information. This provides opportunity for knowledge management tools to capture the knowledge that is shared.

One of the main reasons why agile methodologies reduce the emphasis on documents other than source code is that the cost of creating and, in particular, keeping them up-to-date with the continuously changing requirements and source code (or project state/environment) do not pay off. This maintenance problem of keeping externalized knowledge bases up-to-date was also the reason why many knowledge management approaches failed in the end[1].

However, for any software development project there are a number of information sources that contain useful knowledge and need not be actively maintained by the development team; typical examples are e-mail/newsgroup postings discussing technical issues, lessons-learned stories maintained by a central process group, or web sites maintained either within or outside the company containing material about technologies that are used by the project. Thus, even during development activities that occur within software projects that follow a light-weight process, information is often available that could help team members to successfully perform their tasks.

Since studies have shown that people often are not aware of information that might be relevant to them[2] [8] we are investigating ways to pro-actively provide developers with access to information specific to their current tasks and preferences. In the following, we present a systematic, bottom-up approach on capturing recurrent information needs (including potential ways to satisfy those needs) that typically arise for team members as they are performing software development tasks. Depending on a characterization of their current situation (i.e. current activities, individual preferences and skills etc.), developers are provided with those modeled information needs that are triggered by the characterization; in particular, corresponding information items are retrieved for each of these information needs, which are assumed to satisfy the information needs in required detail.

In order to illustrate this approach, we have constructed a system, called the *Process-oriented Information resource Management Environment* (*PRIME*). PRIME pro-

---

[1] In fact, the knowledge base maintenance problem was never solved in the 1980'ies for expert system approaches, and prevented them from wide-spread adaptation in industry.

[2] While pair programming might increase the chances that at least one of the two developers is aware of relevant information, it does not solve the problem in principle.

vides a technical infrastructure for the task-specific capture of information needs and their distribution to developers, as well as feedback communication.

The remainder of this paper is structured as follows: In Section 2, we discuss existing approaches that help in knowledge sharing in distributed agile teams. Section 3 gives an overview on the functionality provided by PRIME, and presents the concepts underlying our approach. Related work on process-oriented knowledge management is discussed and compared with PRIME in the last two sections.

2 Communities of Practice

Agile methods value people and communication over tools and documentation. In its last consequence, using agile methods implies that knowledge management initiatives need to focus on establishing and supporting Communities of Practice [17]. In a software development environment, communities of practice can loosely be defined as groups of people who work on similar topics, use similar approaches & technologies and have similar information needs. As we concentrate in the paper on the knowledge management aspect of agile teams, the last issue is of most interest to us. Thus, the questions to answer are:

- How can we identify communities of practice?
- What kind of support can we provide to them?
- How can we reduce the knowledge maintenance problem?

In this paper, we address these issues from the specific perspective of virtual agile teams (VAT). An agile team is a software development team whose work practices are inspired by agile methods like Extreme Programming, Scrum, Feature-Driven Development, Adaptive Software Development, Agile Software Development and others. A virtual team consists of geographically dispersed members working on a common goal. Team members either belong to the same company, to a virtual enterprise or to volunteer efforts (e.g. open-source projects). A VAT combines these two aspects. In the following, we deal with the question of how knowledge sharing can be supported in VATs.

2.1 Identifying Communities of Practice

We see several ways how a community of practice can be identified in a VAT setting. First, a team working on the same project implicitly is a community of practice. Team members share several information needs, e.g. what is the state of the work, where can I find information about technologies used in the project etc. Second, technologies and tools used implicitly define a community of practice, e.g. people developing EJB-based applications can share their insights with each other. A similar argument holds for tools used by a group of people. A third category is based on the type of task that a group of people is performing, e.g. software testers from various projects may be able to share knowledge.

Communities of practice based on technology or tool use are orthogonal to project-oriented or task-type-oriented communities. This is immediately visible by looking on various technology or tool oriented newsgroups or web sites where people from different organizations share their questions and answers.

2.2 Supporting Communities of Practice in Virtual Teams

Communities of practice often rely on face-to-face meetings where colleagues exchange knowledge via informal communication. This is exemplified by special interest groups of organizations like IEEE or ACM, IT conferences or quality circles in companies. While agile methods are creating project-oriented communities of practice by stressing pair programming and (often) shared ownership, they are not explicitly tapping into communities based on technologies, tools and task types.

In a virtual environment, communities of practice are based on web sites and/or newsgroups. They can either be centered on technologies (e.g. http://www.theserverside.com/home/index.jsp centers around J2EE development), tools (e.g. http://www.jboss.org/ focuses on the JBoss server), or task types (http://www.testing.com is a web site focusing on software testing). Project-oriented portals are addressing knowledge management issues for distributed teams (e.g. http://sourceforge.net/ host more than 46000 open-source projects).
The following technologies seem to be useful for knowledge management in virtual teams:

- Expert directories: Listings of people and their skill sets that allow finding experts in specific technologies, tools or processes. This can be combined with some kind of evaluation mechanism and subcontracting mechanisms (e.g. http://www.elance.com/ is a marketplace for finding free lance contractors). Within companies, especially consultancy companies, expert directories are often combined with retrieval mechanisms for finding people that worked on specific projects.
- Web sites: Lots of knowledge is freely available on web sites. Web sites often maintain FAQs and cross-list other sources of information for a given topic.
- Newsgroups and mailing lists: While web sites often serve as information providers, they often do not support interactive questioning. Newsgroups, on the other hand, allow a user to ask questions. Often, somebody else from somewhere in the world will post the answer quickly. Newsgroups and mailing lists provide a similar functionality: broadcasting information to many recipients. Newsgroups are following a pull approach: a user has to actively read a newsgroup to find if she can help somebody else. Mailing lists, on the other hand, push questions into the inboxes of all subscribes and bring questions to their attention.
- Information retrieval on the Web: The problem with the Web is to find the gems of relevant knowledge in the vast amount of information available. Search engines and web directories can be of help in that. Search engines use information retrieval algorithms to get high precision and recall on searches. Their limitations stem from the syntactical nature of these algorithms. The semantic web initiative tries to overcome this by using ontologies, inference engines and human modeling effort.
- Collaborative filtering: Communities of practice can be used to determine relevance of information for others. The underlying assumption here is that if two people belong to the same community, they are interested in the same information. Amazon.com, for example, is using this approach for sales support: people browsing to one specific book are shown similar books. Sharing a bookmark list within a community of practice (e.g. a project team) reaches a similar effect.

2.3 Reducing the Knowledge Maintenance Problem

Maintaining knowledge sources in areas were knowledge is changing quickly is a costly undertaking. On the other hand, we argue that there is a vast amount of knowledge freely available and already maintained by some group with an interest in it. Hence, the knowledge maintenance problem can be reduced for a VAT by maintaining the information needs that it has instead of maintaining resources that fulfill these needs. Thus, our approach models the information needs of a VAT and uses existing resources for fulfilling these needs. The reduction in knowledge maintenance effort is based on the assumption that it is much less costly for a VAT to maintain a list of questions than to maintain a list of questions and their answers.

The questions that the VAT maintains can be parameterized. The parameters can be bound before the question is send to a knowledge source with context-specific values. E.g. a question can be: "Give me information about EJB technology provided that the EJB-Skill-Level is ?x and the EJB-Server is ?y". Before the question is send to a knowledge source, the skill level can be bound to "low" and the EJB-Server can be bound to "JBoss". These two pieces of information can be extracted from the current task of the project. Obviously, this requires a model of tasks and task-specific information needs; such a model is described in the remainder of this paper.

3 PRIME

Our approach is based on the assumption that developers are willing to maintain their individual lists of current development tasks (called to-do lists) in an application provided by their company. Tools like our MILOS-ASE[3] support this functionality for virtual agile teams. Tasks are represented by a short textual description, together with some (optional) scheduling information (e.g. a due date). They are assigned to iterations of the agile process.

While developer is working on one of his tasks, certain *information needs* arise for him that need to be satisfied in order to successfully perform the activity[4]. These information needs range from simple questions (e.g. "*How do I launch VisualAge in this company?*") to more questions that usually are more complicated to answer (e.g. "*What issues need to be addressed when using Serialization together with EJB?*").

Even in software organizations that follow a light-weight process, typically there are several information sources available which potentially contain information that can be used to satisfy the employees' information needs. These information sources can be either human subject-matter experts (e.g. experienced colleagues) that employees can contact, or any electronic information system that is accessible by employees. Furthermore, information sources might either be maintained outside the company (e.g. newsgroups, mailing list archives, tool vendor websites, etc.), or they might be internally maintained within the organization (e.g. the company's document management system (DMS), bug tracking systems, lessons learned systems, etc.). For software organizations, the existence of external information sources is an important factor, as a considerable amount of relevant up-to-date technical knowledge (in the

[3] http://sern.ucalgary.ca/~milos

[4] In the following, we use the terms 'task' and 'activity' synonymously.

form of documents, newsgroup postings, etc.) is created and made available outside the organization via Internet technology.

Rather than trying to capture or maintain this knowledge for use within the (virtual) software organization, our approach focuses on connecting developers with knowledge sources that are recommended by those communities of practice addressed by the developer's current task. This functionality has been realized in PRIME, a system to capture, distribute and satisfy task-specific information needs. PRIME has been linked with the project support system MILOS [20], the predecessor of the web-based MILOS-ASE system which has been tailored to support distributed agile software projects.

As a basic functionality, MILOS allows each developer to maintain a basic to-do list (see Fig. 1). In addition, each developer is provided with the PRIME Information Assistant component that enables him to access preferred information items, as well as to initiate an automatic retrieval of information items in order to satisfy an actual information need that was expected to arise for him during his current task. In correspondence to these two concepts, the Information Assistant window is divided into two panes, which are explained in the following two sections.

3.1 Personal Task-Specific Information Needs

The pane labeled 'Private InfoNeeds' allows the developer to associate information items (in the form of bookmarks/favorites) with each task on his to-do list that he considers as useful for this task (see Fig. 1). Furthermore, the PRIME Information Assistant allows users to post task-specific questions or information requests to a forum that is used by all members of the virtual team as a means to support each other by posting answers to a colleague's questions (Fig. 2 & 3). Supporting each other is expected by all members of an agile team and helps building a community spirit. The system creates a link to the corresponding question/answer thread and maintains this link as another task-specific information item, providing the user with immediate access to his postings.

The system stores all tasks together with their associated information items; during future tasks, users can search the set of former tasks and associated links via keyword search on their textual descriptions, in order to make use of formerly found information items during later tasks.

3.2 Recurrent Information Needs

The Information Assistant pane labeled 'Global InfoNeeds' allows the developer to browse the set of recurrent information needs that have been modeled in anticipation of likely information needs that might arise for him during the selected task. In particular, the developer can initiate predefined queries that can be performed on available information sources which are supposed to satisfy one selected information need (cf. Fig. 1).

The set of retrieved information needs depends on the characterization the developer has given for his task so far. In PRIME, a characterization consists of

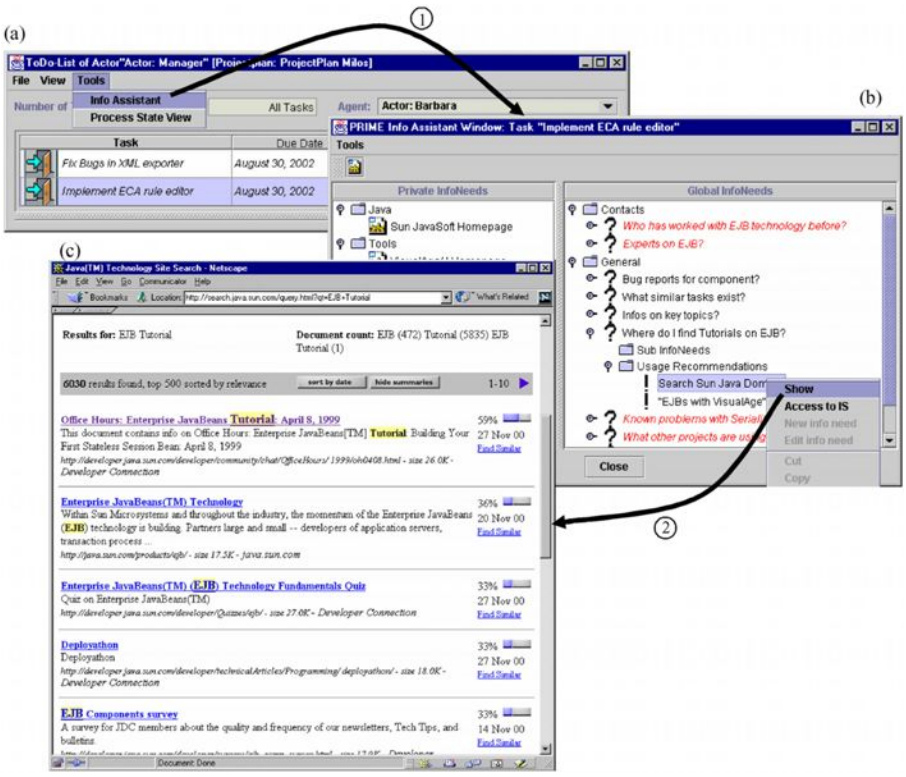

Fig. 1. Snapshot from a to-do list (a), the Information Assistant (b) launched for a selected task (1), and a query execution for an information need to find an EJB tutorial (c): the developer has selected the question "Where can I find a tutorial on EJB?" in the Information Assistant. Issuing the "Show" command on a corresponding IS usage recommendation (2), a browser opens and presents her a list of links which have been retrieved from the Javasoft homepage to the topic "EJB Tutorial". The developer can now refer to the hyperlinks to access the information items.

(i) a classification of the task (i.e. selecting a task type that fits the current task, e.g. “black-box testing”), and
(ii) choosing values for attributes that can be used to further describe the task (e.g. tools or technologies that will be used for executing the task, software components handled during the task, or other key topics that the developer would like to obtain information about).

Figure 4 shows the characterization editor provided by PRIME, which allows developers to characterize their tasks as well as other task-related entities. Task characterizations can be based on entities specified in an organization-specific ontology that is created and maintained by members of different communities that have emerged over time within the virtual organization. Typical entities listed in such an ontology are different task types and tools/technologies, in correspondence to the kinds of communities of practice identified in Section 2.1.

The main purpose of this ontology is to provide each community of practice with the means to systematically define and organize sets of recommended information

resources concerning the entities defined in the ontology. Fig. 5 depicts a simplified excerpt from a task-type hierarchy with associated recurrent information needs.

In the following section, we describe the structure of recurrent information needs in more detail.

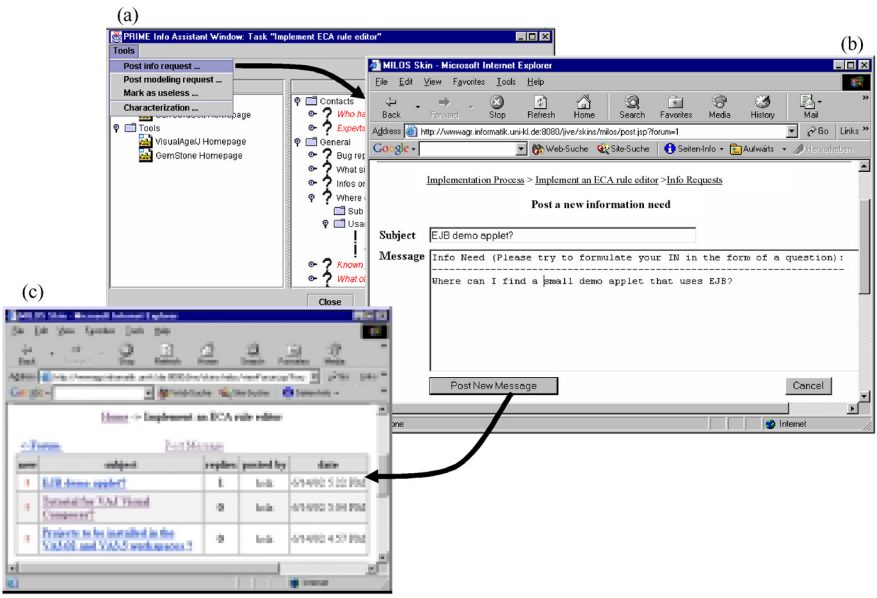

Fig. 2. Snapshot from an agent's Information Assistant (a) that allows agents to post their request (b) directly to a forum (c). For each task, a message forum is provided that maintains the agents' information requests and the replies posted by colleagues. The agents' requests are posted to the corresponding forum by the Information Assistant, after having been extended by a link to the activity during which the information need occurred.

3.3 Representing Recurrent Information Needs

We assume that information is represented as *knowledge items* (or *information items*). A knowledge item is any document (e.g. a MS-Word document, a web page, or an email) that is available to an agent (developer). We say that *information is provided to an agent* if a set of knowledge items is presented to him.

Knowledge items can be obtained by accessing/querying *information sources* that are available to the organization; typical examples of information sources are databases, Document Management Systems (DMS), Web search engines (e.g. Google, AltaVista, etc) or even experienced colleagues. It should be noted that a knowledge item retrieved from an information source can reference another knowledge item or even another information source. An example of such a knowledge item would be an e-mail of a colleague, in which he recommends to consult a particular database.

In the following, we introduce a number of concepts to represent knowledge about available information items that might be useful for agents during activities of a certain class.

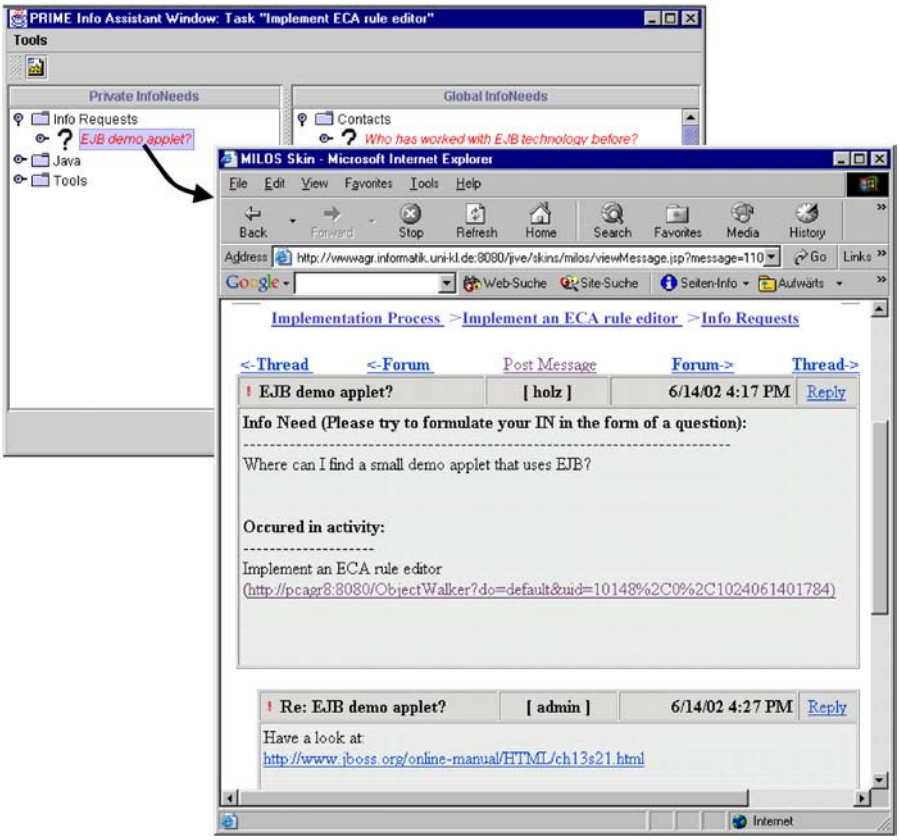

Fig. 3. Snapshot from the Information Assistant for a selected task (a) and thread stored within the task-specific information request forum (b). The Information Assistant maintains a link to the request posted by the agent in the context of a particular task (cf. Fig. 2). Thus, the agent is provided with direct access to the communication thread, i.e. to answers posted by colleagues.

An information source (IS) is represented by the following aspects:

- **name:** the name by which the information source is commonly referred to within the organization
- **contents description:** a short text that explains what information is stored here, and how the information source can be used
- **access:** specifies where the information source can be found or accessed (e.g. a URL in case of an online resource, or contact information about a colleague/expert)
- **query interface:** specifies the information source's interface for automated retrieval, if available (e.g. a CGI script to retrieve items from the information source). This interface will be used to execute queries that have been specified within information needs (see below)
- **quality/cost aspects:** a set of aspects describing quality and cost aspects of the information source (e.g. access cost in case of commercial information services)

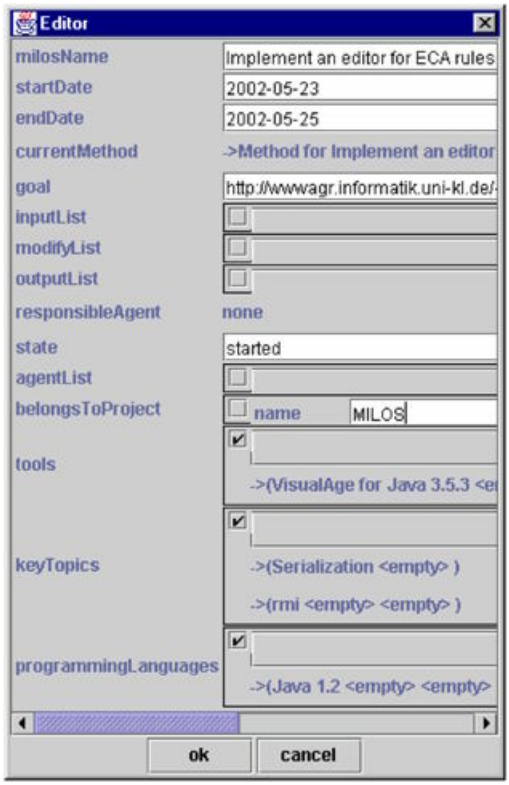

Fig. 4. Snapshot from the characterization editor that allows users to set/change attribute values for a selected task. Here, the user has specified 'VisualAge for Java' as one of the tools that is used during this task, 'Serialization' and 'RMI' as the tasks's key topics, and 'Java 1.2' as the programming language used for coding.

Table 1. Example for an information source representing a Java JDK1.2 language specification document that can be browsed by humans or searched automatically.

IS aspect	**Value**
name	Java JDK1.2 language specification
contents description	Official language specification for Java JDK 1.2
access	http://java.sun.com/products/jdk/1.2/docs/api/
query interface	http://search.java.sun.com/search/java/
quality/cost aspects	high reliability, freely available

Table 1 shows an example for the representation of an information source.

It should be noted that an individual document can also be represented as a special case of an information source (i.e. an information source that contains only one document).

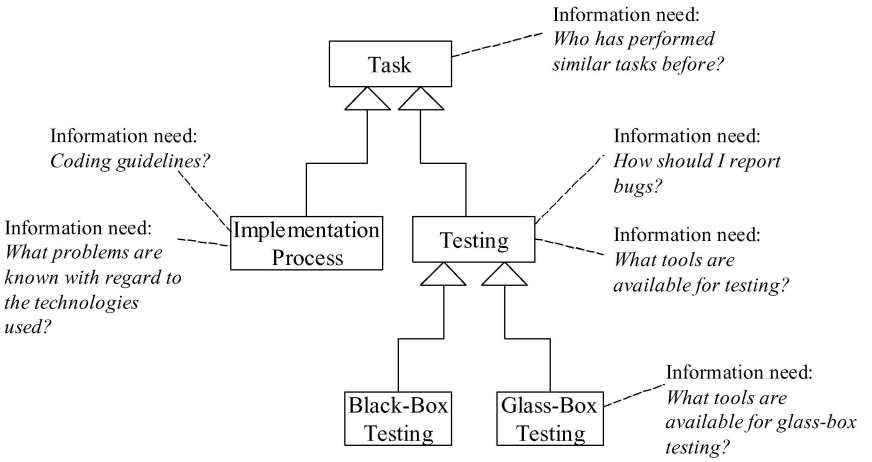

Fig. 5. Excerpt from a task-type specialization hierarchy (depicted as a UML class diagram). Information needs associated with a task type are supposed to be inherited by its sub-types.

Which information sources contain useful information for an agent during his work typically will depend on certain activity and agent characteristics. Furthermore, software engineering activities often undergo changes during their enactment (e.g. schedule changes, product feature changes etc.). As a consequence, the set of information sources that contain useful information changes during an activity's enactment, in correspondence to the activity's changing characteristics.

Consequently, a static list of information sources as a means to provide agents with useful knowledge items is often inadequate for software engineering activities. This leads to the concept of situation-specific *information source recommendations* in order to capture (meta-)knowledge that a certain information source might be useful to agents during activities of a certain type. An IS recommendation is represented by the following aspects:

- **information source:** the information source being recommended
- **task type:** the class of activities for which the information source might contain useful information
- **activity constraints:** specifies conditions on activity characteristics. The information source is only recommended if the conditions hold
- **role constraints:** restrict the recommendation to those agents performing a certain role in the activity
- **skill constraints:** specify conditions concerning the agent's skill profile that must hold in order for the information source to be recommended.

Table 2 shows an example of an information source recommendation.

So far, information source recommendations only describe which information sources are generally considered to be useful during an activity; they do not describe explicitly for what purpose they are considered to be useful, i.e. what information needs might be satisfied by their contents. In order to capture this knowledge, we introduce the concept of information needs.

Table 2. The information source "Java DK1.2 language specification" (see Table 1) is considered useful for all agents taking part in an implementation activity in the role of a "programmer", but only if the implementation language is Java 1.2 and the agent is not already known to be a Java 1.2 expert.

IS recommendation aspect	Value
information source	Java JDK1.2 language specification
task type	Implementation Process
activity constraints	programming language is Java 1.2
role constraints	useful for role 'programmer'
skill constraints	programmer is not a Java 1.2 expert

An information need (IN) encompasses a situation where an agent requires certain information in order to successfully carry out a given activity. We assume that information needs are being expressed in form of a question (e.g. "*Where can I find a tutorial on EJB?*"). These questions are supposed to be of the kind "*Where can I find pieces of information on ..., because it might help me to solve problem x?*", rather than "*What is the solution to problem x?*". In that way, information needs describe goals that, when achieved, enable agents to successfully perform their activities, which in turn are intended to achieve a certain project objectives.

Whether a certain information need arises for an agent will depend on certain activity and agent characteristics (e.g. the technologies that have to be used, the agents experience, skills etc.). Hence, a captured, expected information need should include a specification of the situations in which they typically occur, as introduced for the capture of information source recommendation.

Information needs potentially can be satisfied by accessing one or more information sources via their interface (e.g. send e-mail to a colleague, launch a tool to open a document, or query an information system). As a result, the information source returns one or more information items (e.g. a human answers by e-mail or the Document Management System returns a set of documents). The interpretation of these information items is supposed to either satisfy the information need directly, or help to satisfy it by referring to another information source that might contain the information required to satisfy the information need.

In order to provide a template for the description of a way to access an information source to potentially satisfy an information need under certain conditions, we introduce the concept of an *information source usage recommendation* (*IS usage recommendation*). IS usage recommendations are represented by the following aspects:

- **information source recommendation:** specifies the information source that
- potentially contains information to satisfy an information need, as well as the conditions (in terms of activity, skill and role constraints) when the information source is recommended to be accessed to satisfy the information need.
- **usage direction:** either is a short text explaining in natural language where to find the desired information, or it is a query specification. In the latter case, the query is specified by the following aspects:
  - **comment:** is a short text explaining the query's semantics to the human reader.
  - **queryCommand:** contains a query expression that can be sent to the information source via its query interface (see above).

Building on the concepts introduced so far, a recurrent information need is represented by the following aspects:

- **question:** a textual representation that describes the information need.
- **information source usage recommendations:** a list of information source usage
- recommendations, describing alternative ways to potentially satisfy the information
- need under certain conditions.
- **task type:** the class of activities during which the information need is expected to arise.
- **activity constraints:** specifies conditions on activity characteristics. The information need is only expected to arise if the conditions hold
- **role constraints:** describes for which roles the information need is expected to arise.
- **skill constraints:** specifies conditions concerning the skill profile of an agent participating in the activity. The information need is only expected to arise if the conditions hold.
- **sub-information needs:** references a set of sub-information needs; satisfying these information needs is assumed to provide information that helps in satisfying the referencing "parent" information need.

Figure 6 shows a screenshot from the Information Need Manager interface to define information needs.

We assume that the definition of recurrent information needs will be triggered by either of the following situations:

1. A subject-matter expert posted an answer to an information need, but still finds himself being repeatedly asked to answer this question again by different colleagues. Consequently, he would like to have users being automatically referred to the already documented question/answer thread.
2. Instead of searching the set of former tasks in order to find useful information items for his current task, users might request to have certain information items offered to them on a regular basis (e.g. “Offer me the Java Language Spec. whenever I perform an implementation task that includes coding in Java”).

Figure 7 summarizes the relationships between the different constructs introduced above.

Information source recommendations and information needs are used to differentiate conceptually between two strategies:

- providing access to an information item or source, without stating explicitly what information need(s) it is intended to satisfy
- presenting explicitly formulated information needs in form of a question, together with information items that potentially provide answers to the question. Thus, the question denotes the purpose for offering the information items.

In summary, information source recommendations are used whenever

(i) it is obvious what the corresponding information source is used for (e.g., a language specification will always be used for reference), or

(ii) there are so many different ways of usage that it would be too cumbersome to explicitly list all of them.

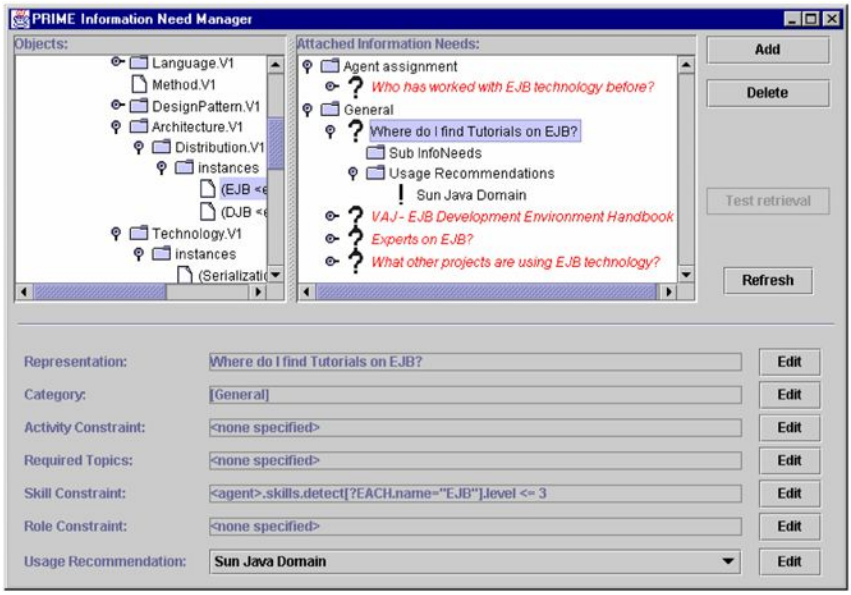

Fig. 6. Snapshot from the Information Need Manager interface: from the tree in the upper-left part of the window, the user has selected the entity 'EJB' from the domain ontology. The tree in the upper-right part displays the information needs associated with that entity, grouped under appropriate categories (rendered as folders). The attribute values of the selected information need are shown in the lower part of the window. For example, the skill constraint is shown, formalizing that the selected information need should only be offered to developers whose skill level concerning 'EJB' technology the less than or equal 3.

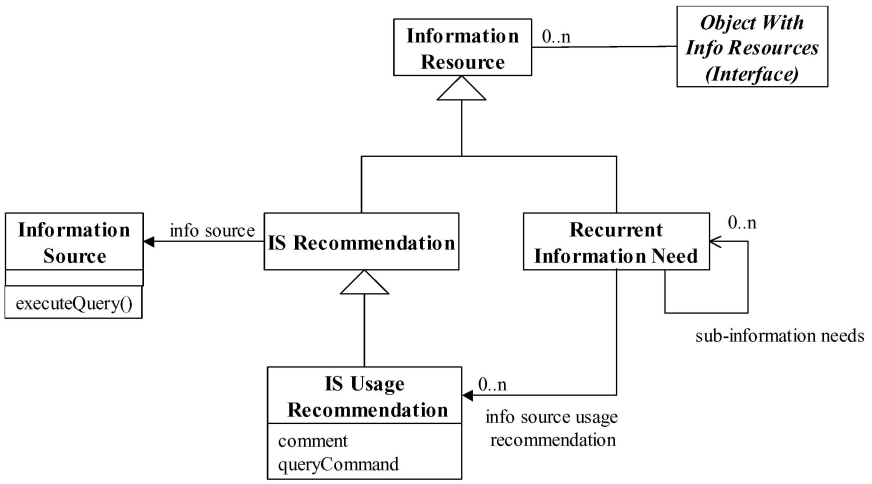

Fig. 7. UML class diagram depicting the relations between the concepts introduced to represent recurrent information needs.

In contrast, the representation of information needs allows capturing in more detail

- *what* information might be useful (expressed as a question)
- *where* and *how* this information can be found (i.e. a list of information sources that potentially contain the information, together with direction on how to access them)
- *when* it might be useful (i.e. constraints on certain activity characteristics available at enactment time)*when* it might be useful (i.e. constraints on certain activity characteristics available at enactment time)
- *to whom* it might be useful (i.e. constraints on performers' roles and skills).

4 Related Work

Most work on integrating Knowledge Management and process support has been done in the field of business processes (see [3] for a recent overview). In the following, we discuss and compare PRIME to related state-of-the-art approaches.

TIDE [18] is a web-based system that facilitates task-based retrieval of documents. A task in TIDE describes a yes/no question that a user is trying to answer; it is represented by a set of weighted slot/value pairs that characterize the question, where all values are terms (words or word stems). The weight of an attribute "loosely ... represents the importance or frequency of the value of that slot in relevant documents." [18]. Furthermore, a task (question) references a set of sub-questions that provide evidence towards answering the parent. This task hierarchy "corresponds to a Bayesian Network, which encodes the probabilistic relationships between questions" [18]. The weights and the task hierarchy are maintained in a task model that users instantiate for their concrete tasks.

The task representation is used to retrieve task-specific relevant documents via the vector space model [12]. Each document is characterized by a vector of weighted terms. A weighted keyword query is derived from a user's task representation by recursively collecting the terms from the question's sub-questions and computing weights for these terms according to the sub-questions' importance to the parent question. TIDE's method of query derivation allows the relevance criterion to be adapted to reflect changes in the users' opinion on relevance by weight modifications.

Compared to PRIME, two main differences can be identified: first, TIDE restricts the notion of a user's task to answering yes/no questions, whereas activities in PRIME reflect arbitrary tasks. Second, the determination of relevance in TIDE is computed by a weighted term query approach; in PRIME, relevance computation is two-step process: (i) determination of relevant information needs based on a symbolic activate/trigger model[5] using boolean expressions, and (ii) launching a well-formed query command to an appropriate information source as defined by the information needs. Thus, in TIDE relevance can only be expressed heuristically in terms of a probabilistic model; in PRIME, relevance can be formulated as a logical fact, which allows it to enforce that certain important documents are always retrieved in specific situations. Also, TIDE's weight-based relevance model will be difficult to maintain, as it is not trivial to identify which weights have to be changed in what way for an intended up-

[5] Information resources are activated by entities present in the activity representation; activated information resource trigger if their constraints are satisfied. Only triggered resources are presented to the agent performing the activity.

date of the relevance criterion. However, the TIDE system might be an interesting extension to PRIME, as TIDE's notion of a task actually corresponds more closely to PRIME's notion of an information need (formulated as a yes/no question): by mapping TIDE's tasks to PRIME's information needs, the information need's query command could be used to trigger TIDE's retrieval mechanism.

EULE [9] is a system that provides computer-based guidance for office workers at Swiss Life. It introduces a formal knowledge representation language that covers data and process aspects, as well as legislation and company regulations relevant for office tasks dealing with life insurance. Users are guided through a sequence of activities to perform their tasks and are being provided with access to relevant documents (contracts, letters, client data etc.); for each activity, users are requested to enter taskspecific data into forms that are presented to them by EULE. Depending on the data entered, new activities might be triggered because of certain laws or regulations. EULE uses deduction to create appropriate instances of rights and obligations, which are represented as concepts; its inference engine couples description logic and deductive database technology.

For each activity, EULE can present an explanation to the user why the activity has to be performed. Furthermore, letters that have to be created during certain activities can be generated automatically from the user's data (in combination with the company's databases). The system was introduced at Swiss Life in mid-1999, and is reported to be highly accepted by employees. Perhaps most interestingly for the approach presented in this thesis, a field study with EULE has been conducted at Swiss Life with positive results: team heads "noticed a considerable relief from the support they usually need to give their team members whenever they encounter a situation they do not know how to deal with" [9].

Because of its inflexible workflow enactment model (build/compile/execute lifecycle), EULE is inadequate to support software development processes[6]. Furthermore, EULE is not designed to provide users with information from external information sources. In addition to the documents related to an activity (contracts, letters, etc.), users are given access to textual representations of laws and regulations that are relevant to the user's current activity. In particular, relevance of information is determined strictly deductively in EULE. In PRIME, relevance can also be computed deductively (by means of information need preconditions formalized in F-Logic); but additionally, information can be retrieved via soft-matching mechanisms (e.g. standard information retrieval approaches [12], similarity measures [11] etc.). Depending on the query command specified within an information need and the retrieval mechanisms supported by available information sources, soft-matching can be used to find relevant information whenever this seems appropriate.

Schnurr et. al. describe an approach based on OntoBroker[7] for Reactive Agent Support [13, 14]. OntoBroker is used to define a domain ontology and to manage an archive of ontology-annotated documents. In addition, OntoBroker scans the documents for facts, stores them in a database, and infers facts from the database using a built-in inference engine.

In OntoBroker, business processes are defined as SGML nets (a special kind of Petri nets). Processes are represented by transitions; predicates based on document contents define when a transition may be executed. Queries (called context-based

---

[6] In fact, EULE is a single-user system and does not support workflows performed by project teams.

[7] OntoBroker is a commercially available F-Logic interpreter (www.ontoprise.com).

views) to the database can be associated to transitions and places. The approach focuses on strongly-structured processes, as the planning of activities and remodelling of SGML nets is not supported. Furthermore, the approach is restricted to F-Logic-based queries to one central repository of annotated documents. For PRIME, F-Logic-based queries to an OntoBroker repository only form one of many possibilities to retrieve information; alternatively, it can provide information retrieved from standard information retrieval systems, relational databases, or case-based reasoning systems. Especially the latter are considered to be of prime importance for experience management within software organizations [15].

Furthermore, the proposed SGML net-based approach does not facilitate an explicit representation of activities. Rather, the activity states are implicitly defined by the state of document attributes. As a consequence, queries can only reference attributes of the document currently being modified by an activity (i.e. transition).

In [16], Wargitsch et. al. present the OM-based flexible WFMS WorkBrain that provides integrated access to different information and communication services. These include a CBR system (storing former workflow cases), a workflow issuebased information system (WIBIS), a mechanical design database, an electronic product catalogue, a know-how database for engineering solutions as well as a traditional DMS.

WorkBrain supports both structural planning and enactment tasks: e.g. workflow construction is supported by retrieving similar former workflow cases, whereas enactment tasks are supported by retrieving documents created in former workflows. However, only the WIBIS system is process-oriented in the sense that processes are used to organize issue threads. Access to the other information systems can not be tailored to specific tasks; in particular, generic queries that are instantiated for concrete tasks cannot be modeled. While the OM is comprised of different information sources that have been made available, no process-specific usage is supported and no automatic query execution takes place, i.e. the OMIS is passive.

The KnowMore framework [2] outlines a three-step deployment process for their workflow-enabled information delivery system. First, a commercial business process modeling tool is used to define a process representation that can be enacted by a workflow engine. Second, knowledge-intensive tasks (KIT) within this process model are identified; these are enriched with KIT variables and with conditional, generic queries. KIT variables represent slots that have to be filled during process enactment, whereas queries represent potential information needs.

During workflow enactment, the generic queries are instantiated with workflow parameters in the context of concrete tasks. After instantiation, the queries are executed by computer agents which encapsulate knowledge on how to retrieve information from a particular information source. The results can automatically be integrated into document templates that specify the input fields that have be filled with retrieved information. In addition, KnowMore users can be presented with explanatory information on the values chosen/retrieved for the template's input fields. The retrieval results are updated whenever the context in which they have been retrieved changes.

Like OntoBroker/SGML, the KnowMore approach focuses on strongly-structured processes and the automated integration of retrieved information, both of which are inadequate for software development processes. With KnowMore, only the enactors (but not the planners) of workflows are supported by the automated information retrieval, and the set of information needs is defined statically in the process model.

KnowMore and PRIME also differ in their main strategy for knowledge delivery. The KnowMore system always automatically executes the whole set of information needs currently regarded as relevant, and then post-processes the results. In PRIME, the agent is given the possibility to choose from a set of offered information needs the one that she considers as relevant in her current situation. The approach implemented in KnowMore is indented to support (automatically) filling in the structured document template, whereas PRIME is intended to support creative processes handling informally specified documents. In particular, the objective of information needs in PRIME is not to fill in the slots (i.e. attributes) of a document's characterization object. On the contrary, the attributed are used to retrieve information items that help a human agent to successfully perform a creative activity.

DECOR[8] [1] builds upon the KnowMore framework, but addresses weaklystructured, knowledge-intensive processes which can not be planned fully in advance. Similar to PRIME, an Information Assistant is proposed that observes the workflow and interprets modelled information needs specified in the process model in order to offer relevant information. The main focus of the DECOR project is to provide a practice-driven, "total solution" for the integration of information retrieval into workflow-embedded, knowledge-intensive tasks. To this end, the project utilizes available, consolidated modeling methods and information technology in combination with research results from the KnowMore approach. However, continuous information need evolution as facilitated with PRIME is not reported to be addressed by DECOR.

Another system that shares some similarities with PRIME is Answer Garden 2 (AG2) [19], which supports astrophysicists in data analysis tasks. AG2 provides an integrated interface that allows users to locate and use about one thousand software components, their associated documentation, tutorials, frequently asked questions, data analysis recipes, or to ask a specific scientific community for help. It relieves users of the burden to remember the different data analysis tools, data formats, interfaces, and help systems, and provides shared recipes on how to use them. In particular, AG2 facilitates the collection and dissemination of organizational knowledge by building a database of commonly asked questions that 'grows "organically" as new questions arise and are answered' [19]. However, AG2 does not allow for different types of tasks, explicit task characterizations, or proactive, situation-specific distribution of those commonly asked questions.

5 Conclusion

Whereas face-to-face communication between team members might have the highest bandwidth for knowledge exchange, there are circumstances when this is either not feasible (as e.g. for VATs) or not always desirable (e.g. because of a communication overload for experts). In addition, a considerable amount of explicit knowledge is available on the Internet in the form of newsgroup postings, technology reports, web sites dedicated to certain tools/technologies, etc. Hence, additional support should be made available to team members to promote the use of information sources that are readily available.

A similar situation appears in open source projects, where newcomers face the problem of catching up with the knowledge of experienced project members, part of

[8] Delivery of context-sensitive organizational knowledge.

which is reflected in mailing list archives. We believe that capturing and distributing proactively typical information needs of newcomers with regard to certain system components, technology used, etc. will greatly relieve experienced group members from having to answer the same standard questions repeatedly; at the same time, newcomers are relieved from sifting through large FAQ and mailing lists.

In this paper, we presented a system to capture and distribute task-specific knowledge about available information resources in the form of explicitly represented recurrent information needs. Depending on the characterization of a currently selected task, a set of information needs is retrieved and presented to the user in the form of a list of corresponding textual questions. From this list, the team member is assumed to choose one that corresponds best to his current information need. For this chosen information need, the predefined information source usage recommendations are executed (i.e. the specified query commands are instantiated and sent to appropriate information systems, or contact information for human subject-matter experts is displayed) to provide the user with information items that potentially satisfy his information need.

In particular, PRIME allows a smooth introduction of Knowledge Management services into the every-day work practice of members of a VAT. To begin with, the system can be used by team members to maintain task-specific bookmarks (i.e. URL links to favourite documents), providing users with a task-oriented way to organize and quickly access their documents. As valuable information is already available on the Internet or in the organization's document repository, we argue that our approach ameliorates the knowledge acquisition bottleneck problem that let many KM initiatives fail in the beginning. In addition, PRIME's forum component[9] serves as platform for task-specific communication with the experienced colleagues. That way, additional task-specific information items can be captured on the fly.

Up to this stage, no modelling of a domain ontology is required. Furthermore, initial modelling of recurrent information need can start from basic task characterizations that consist of keyword lists to name any tools and technologies handled during the task; corresponding query commands can then search the available information sources for appropriate keyword (combinations). Only when

- users start to express an interest in being provided with certain bookmarks on a systematic basis during a certain class of activities, or
- subject-matter experts (or community-of-practice members) decide because of repeatedly asked questions that users should be provided with certain information during a class of activities, or whenever they are handling a certain tool/technology,

the need arises to capture and formalize these requests in the form of a corresponding entity in the domain ontology together with a set of explicit information resources, and to provide access to the requested information from appropriate information sources. Because of the personal gain achievable by explicitly modelled and automatically retrieved information needs, we argue that people will be willing to invest some time in modelling efforts (or posting modelling requests). Essentially, the explicitly represented information sources proposed in this work can be seen as a special kind of "markers and props" described in Cockburn's Manifesto For Software Development [6], which people use to "inform, remind and inspire themselves and each

[9] Based on Jive (www.jivesoftware.com)

other in getting the next move on the game", and which serve to "inform and assist the players of the next game".

Currently, the implementation of PRIME has reached a stage where students have started to use it during their implementation activities on MILOS. From our experience gained so far, future extensions of the work presented here will need to address the effort required for information needs modelling. As an alternative to the explicit representation of logical preconditions, we intend to adapt and integrate techniques known from Collaborative Filtering (see e.g. [7]) or Case-Based Reasoning (see e.g. [11, 10]) with our mechanism for situation-specific information need retrieval. Usage of this technology could provide team members with information items that colleagues found useful who had 'similar' information needs, or with former information needs that (other) team members had during 'similar' situations. It is to be hoped that such extensions could narrow the current gap between the low effort required to maintain the users' personally preferred information resources during their activities, and the relatively high effort required to model fully specified, generic information needs. One of the graduate students in Calgary is currently working on a comparison between text retrieval approaches and the ontology-based approach presented here.

Acknowledgements

The work on MILOS was supported by the DFG (as part of SFB 501: "Development of large systems with generic methods"), NSERC, and The University of Calgary, with several research grants. We would like to thank Empolis knowledge management division[10] for providing us with their CBR middleware 'Orenge'.

References

1. A. Abecker, A. Bernardi, S. Ntioudis, G. Mentzas, R. Herterich, C. Houy, S. Müller and M. Legal. The DECOR Toolbox for Workflow-Embedded Organizational Memory Access, In: Proc. of the 3rd Int. Conf. on Enterprise Information Systems (ICEIS 2001), Portugal, July 7-10 2001, Vol. 1, 2001.
2. A. Abecker, A. Bernardi and M. Sintek. Enterprise Information Infrastructures For Active, Context-Specific Knowledge Delivery. ECIS'99 - The 7th European Conference on Information Systems, Copenhagen, Denmark, June 1999.
3. A. Abecker, K. Hinkelmann, H. Maus and H.-J. Müller (Hrsg.). Geschäftsprozessorientiertes Wissensmanagement, Springer, 2002.
4. Beck, K.: Extreme Programming Explained: Embrace Exchange. Addison-Wesley, 1999.
5. Cockburn, A.: Agile Software Development Joins the "Would-Be" Crowd. Cutter IT Journal, Vol. 15, No. 1 (2002) 6-12.
6. Cockburn, A.: Human's and Technology Manifesto For Software Development. http://member.aol.com/acockburn/manifesto.html.
7. H. Liebermann. Letizia: An Agent that assists web browsing. In Proc. of the 13th Int. Joint Conference on Artificial Intelligence, San Francisco, CA, Morgan Kaufmann, 1995.
8. Mahe, S., Rieu, C.: Towards a Pull-Approach of KM for Improving Enterprise Flexibility Responsiveness: A Necessary First Step for Introducing Knowledge Management in Small and Medium Enterprises. In: Proceedings of the International Symposium on Management of Industrial and Corporate Knowledge (ISMICK '97), Compiegne, 1997.

[10] www.empolis.com

9. U. Reimer, A. Margelisch and M. Staudt. A Knowledge-Based Approach to Support Business Processes, AAAI Workshop on Bringing Knowledge to Business Processes, 20-22 March, 2000.
10. M. M. Richter. CBR: Past and Future - A Personal View. Invited Talk, International Conference on Case-Based Reasoning (ICCBR-2001), Vancouver, British Columbia, Canada, 30 July - 2 August 2001. http://wwwagr.informatik.uni-kl.de/~richter/
11. M. M. Richter and K.-D. Althoff. Similarity and Utility in Non-Numerical Domains, Mathematische Methoden der Wirtschaftswissenschaften, Physika-Verlag, pp. 403–413, 2001
12. G. Salton and M. McGill. Introduction to Modern Information Retrieval, McGray-Hill, 1983.
13. H.-P. Schnurr, S. Staab and R. Studer. Ontology-based Process Support. Workshop on Exploring Synergies of Knowledge Management and Case-Based Reasoning (AAAI-99). Technical Report, Menlo Park: AAAI.
14. S. Staab and H.-P. Schnurr. Smart Task Support through Proactive Access to Organizational Memory. Journal of Knowledge-based Systems 13(5). Elsevier, 2000.
15. C. Tautz. Customizing Software Engineering Experience Management Systems to Organizational Needs, Ph.D. thesis, Universität Kaiserslautern, 2000.
16. C. Wargitsch, T. Wewers and F. Theisinger. An Organizational-Memory-Based Approach for an Evolutionary Workflow Management System - Concepts and Implementation, Proceedings of the 31st Annual Hawaii International Conference on System Sciences, 1998, p174-183.
17. Wenger, E., Snyder., W.M.: Communities of Practice: The Organizational Frontier. Harvard Business Review , Jan-Feb (2000) 139-145.
18. M. Wolverton. Task-Based Information Management, ACM Computing Surveys, Vol. 31, Number 2es, 1999.
19. Ackerman, M.S., McDonald, D.W., "Answer Garden 2: Merging Organizational Memory with Collaborative Help." Computer Supported Cooperative Work (CSCW 96), (Boston, MA, 1996), 1996, pp. 97-105.
20. Maurer, F., Dellen, B, Bendeck, F., Goldmann, S., Holz, H., Kötting, B., Schaaf, M.: Merging Project Planning and Web-Enabled Dynamic Workflow Technologies. IEEE Internet Computing May/June 2000, pp. 65-74.

Learning and Understanding a Software Process through Simulation of Its Underlying Model

Holger Neu and Ulrike Becker-Kornstaedt

Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, D-67661 Kaiserslautern
{neu,becker}@iese.fraunhofer.de

Abstract. This paper describes the usage process simulation models for better understanding the dynamic effects of processes. First, a method to build a comprehensive descriptive model is presented, which can build the foundation for the simulation model. Based upon the understanding of the static process, parts of the dynamic model can be modeled can be created. The initial method presented was used to create a discrete event model for inspections. The paper discusses the benefits of simulation for learning and optimizing the feedback cycles for learning.

1 Introduction

Improving an existing process requires capturing the process in the form of a process model. [Dow93]. The models are a prerequisite for developing and producing software with predictable quality and effort. However, the models used and required often only show the static view of the process. Developing software is a dynamic human-based process and this dynamic view is typically not represented in process models. Process changes often have undesired effects and may lead to worse process performance at high cost Especially for new processes it is difficult to determine effects if the process is changed. Process simulation can support the decision for change before the change is actually implemented and money is spent. By "playing" with the model Project Managers and Process Performers, i.e., the people performing the process learn more about the dynamic of the process and can identify false believes and evaluate different alternatives.

Chapter 2 describes the motivation for descriptive process modeling and simulation. Chapter 3 describes a method for developing a descriptive process model and gives information about simulating a software process. Possible steps how to create a simulation model are shown in chapter 4. A short overview of the related work simulation and training or education is presented in chapter 5. In Chapter 6 we discuss the benefits of simulation and the pitfalls to avoid.

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 81–93, 2003.

2 Background

The quality of the process used to develop and maintain a software product has a large impact on the quality of the software product [PWCC95, CG98]. Minor changes to a process that had run smoothly up to now can have a large impact on the resulting product outcome. However, in industrial practice, processes are often changed at will without considering the impact such a process change may have.

Especially in domains where little experience regarding the process exists, overall cause-relationships between the process and the resulting product quality are not always clear. Obtaining a detailed understanding of these relationships would require performing several dry runs of the process under different conditions and recording the results to determine a better understanding of the parameters influencing the process outcome. In practice, this is not feasible, because it is time-intensive and cost-intensive, and involves a high number of staff on experimental processes that do not produce marketable results.

Nowadays many application domains, such as mobile service development or Internet domain, require processes that are extremely flexible. Agile processes are more and more employed in industry because of their high flexibility. This flexibility is needed to cope with constraints of the development environment, such as an underlying technology that is rapidly changing. On the other hand, using these processes encompasses a high risk, because minor changes in the process may lead to a huge loss of quality of the resulting product.

Thus, a mechanism is needed that allows to assess the cause-effect relationships between process changes and the resulting product outcome with little personnel effort, at low cost, and within a reasonable time frame.

A way to assess these cause-effect relationships at low cost, with effort involved from the staff involved, and within a reasonable time frame is the simulation of software processes.

Simulation models are used in many domains to test assumptions and believes of the system modeled. Main perspectives of simulation are better understanding and learning. In the software engineering domain simulation is not as spread as in other domains but organizations start to use simulation for aiding their process understanding and improving. [KMR99] clustered the reasons why a simulation model should be created into six categories.

- Strategic management,
- Planning,
- Control and operational management,
- Process improvement and technology adoption,
- Understanding, and
- Training and learning

To enable long-time learning and improvement in software organizations, process improvement and technology adoption and understanding are the two main objectives. The simulation used for training and learning addresses more the education of students and project personnel.

[KMR99] also explain the possible scope of a simulation model (part of the process or complete process, one project or multiple projects), result variables (effort, cost/time, and defects) and process abstractions (which key activities, resources, primary objects and dependencies) that are useful to consider. Another dimension is the simulation technique that should be used and the data that is possible to obtain. If sufficient data from previous projects is available, calibration and validation can be done based on that data.

To define the scope with a more structured approach [Pfa01] developed a GQM based approach with the five dimensions: Role, Scope, Purpose, Dynamic Focus, and Environment.

3 Method

3.1 Overview

This section describes the overall method to build simulation models in order to increase process understanding and promote a better understanding of the impact modifications to the process may have. Figure 1 depicts the three main steps of the method. In particular, the method consists of two basic phases, *descriptive process modeling* (DPM), resulting in a static model of the process as it takes place, and *process simulation*. Process simulation itself consists of two steps, first a simulation model is developed from the descriptive process model developed in the first step. Additional information relevant for the development and calibration of the simulation model has to be gained through e.g., interviews or measurement data. In a second step the actual process simulation takes place to yield dynamic information on the process.

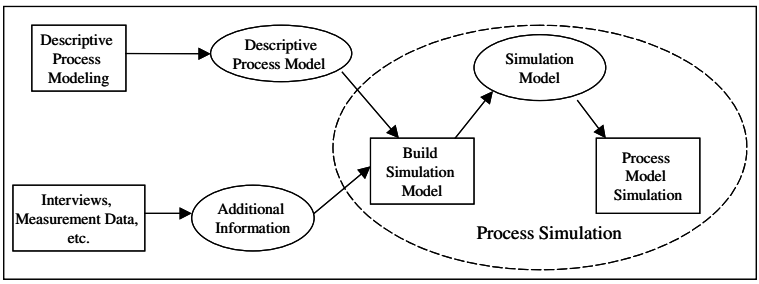

Fig. 1. Main steps

This simulation model can then be used to experiment with variations or modifications of the process without affecting the performance of the process actually performed. The next two sections give background on software process modeling and on simulation of process models.

3.2 Descriptive Software Process Modeling

Descriptive process modeling is concerned with developing a description of the process as it takes place in reality. This section describes the Prospect [Bec01] approach to software process elicitation for descriptive process modeling.

In Prospect, process elicitation is driven by the process-modeling goal – a specification of the process model that is to be developed by the process modeling activity. Using Basili's GQM approach [BCR94] the process-modeling goal can be characterized along the following four dimensions:

- Scope of Study: What processes or what part of the process are to be described by the process model (coverage), and what level of granularity (granularity) is necessary for the process model?
- What will the process model be used for?
- Quality Focus: Which particular aspects should the process focus on, what information is to be depicted in the process model? Based on the quality focus the schema elements that are needed for the process model can be defined.
- Context of study: In what target environment should the process model be valid?

Based on those requirements the schema elements that are needed for the process model can be defined. In the context of Prospect, process information will be characterized with the help of the process-modeling schema described in [WB97], which has been implemented in the Spearmint tool [BHK+99].

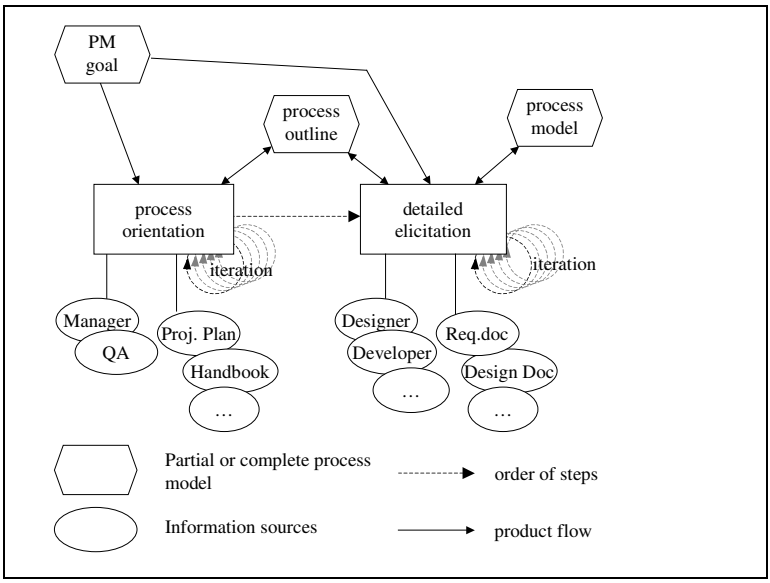

Fig. 2. Prospect activities and main products

Prospect is grouped into two distinct phases, *process orientation*, resulting in the outline of the process, and *detailed elicitation*, resulting in the final process model.

Typical sources for information in the orientation phase are process documentation or project plans and unstructured interviews with management roles. Documents such as project plans or process handbooks list the activity names, dependencies between activities, sometimes artifacts. This information will later facilitate understanding of the terminology used in the organization, and give orientation on whom to interview. Typical human information sources in the orientation stage are Management roles [FFHM00]. These typically have an overview of the process. Documents helpful at this stage include process handbooks or project plans, which give an overview of the process.

In the second phase, *detailed elicitation*, the Process Engineer tries to obtain details of the process. This includes detailed descriptions of the activities, artifacts, roles, and tools modeled in the process, entry or exit criteria for the activities performed, or information on structure and contents of the artifacts used or produced.

Techniques employed in this stage are mostly techniques that allow for obtaining in-depth information.

Subsequently, the model is validated (not part of the Prospect method). Typically, people who provided information for the model are asked to review the model. Structural inconsistencies can be detected with the help of a tool – in case the model was created using a tool. Errors detected may lead to rework and corrections of the model.

Making this process model available to Process Performers, for instance, as electronic process handbooks [KBR+98] on the organization's intra net has the long-term benefit that the process model can be used for learning or improvement. The process model incorporates experience related to process execution. However, this experience is mainly related to static properties.

3.3 Simulation of Process Models

A simulation model can aid multiple purposes through a better understanding of the process. The static information of the process can be captured in the descriptive process model. To elicit the dynamic model for a specific part of the process the descriptive model is a good starting point. For defining the scope of the dynamic model, it is crucial to know the purpose or the questions that should be answered with the simulation model. Due to the complexity of most software processes, it is impossible to create a model that can be used to answer all questions or to capture the complete process with all its details. Defining the scope and the detail of the simulation model can be essential for the usefulness of the simulation model.

Up to now a lot of simulation techniques have been used, among them state based process models, rule based languages, petri nets, continuous, and discrete even simulation. We will address continuous simulation (system dynamics) and discrete event simulation, in the following. System dynamics is the approach that was used for the most models documented in the literature. In literature, some hybrid models are documented (a combination of two techniques, usually system dynamics with discrete event simulation), and only one that uses pure discrete event simulation [ChS00].

A good overview of the advantages and disadvantages of system dynamics models and discrete event models can be found in the article [MaR00]. System dynamics in

general is more suited for models that have a larger scope and address the stability of the process and the levels of people's experience, fatigue, schedule pressure and exhaustion. In addition, in the system dynamics approach objects in the model are not distinguishable from each other and exist only as a numerical value that is divided in fractions during simulation (e.g., 100 code units are just a number and not 100 individual units). For these reasons the size and other attributes of items, or abilities of the participants are usually modeled by average values in system dynamic models (e.g., the five participants have the same productivity and the code unit have all the same size).

In a discrete event model it is easier to model a sequence of steps and to characterize the objects, called items, flowing through the model. Therefore, discrete event simulation is extensively used in the domain of modeling manufacturing or logistic systems. For software processes the items flowing through the model usually represent people, code, defects, rooms. Each of these items can have attributes that capture individual values for this attribute. These attributes describe the abilities of participants or the size of a code unit. In addition, the values of the items' attributes can be sampled from a distribution and capture the uncertainty in measuring the attributes.

Depending on the purpose of the model it can be used for an a priori analysis, "What if" games for changed parts of the process, techniques or changed parameters (e.g., different staffing) or an a posteriori analysis to understand the process and what happened. In all cases, the dynamic models help to learn more about the behavior of the model.

4 Application

For the first models we focus on the analytical processes in the software development process, especially the inspection and testing processes. These processes are important in order to build high quality products. Also they are suited for building a simulation model, because a large amount of empirical data and knowledge exists for this processes. To start the development of the model we built a descriptive model of the inspection process with the activities planning, preparation meeting, preparation, inspection meeting, rework and follow up with the corresponding roles performing these activities. The descriptive model is shown in Figure 3. The five roles in the figure are described in detail in [EbS93].

Before we can build the simulation model, the influencing factors have to be identified. We will apply parts of the System Dynamics technique, the cause effect diagrams, also known as casual loop diagrams [Ste00]. This cause effect diagrams are used for analyzing the relationships of the attributes in the inspection process.

In [LaD89] the authors differentiate the core inspection concepts and relationships into five dimensions. This description is used as a starting point to identify potential impact factors on the variables of interest (e.g., effort, duration, and number of detected defects). These are the interesting values for a simulation that are usually not captured in a static process model. Starting with this information and interviews with experts a cause effect diagram was created (Fig. 4.).

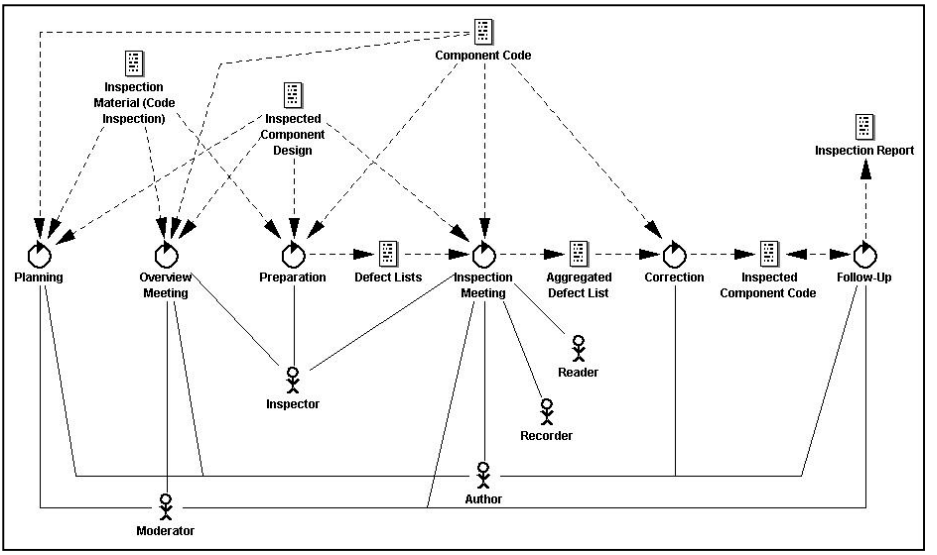

Fig. 3. Detailed product flow of an inspection process of code units

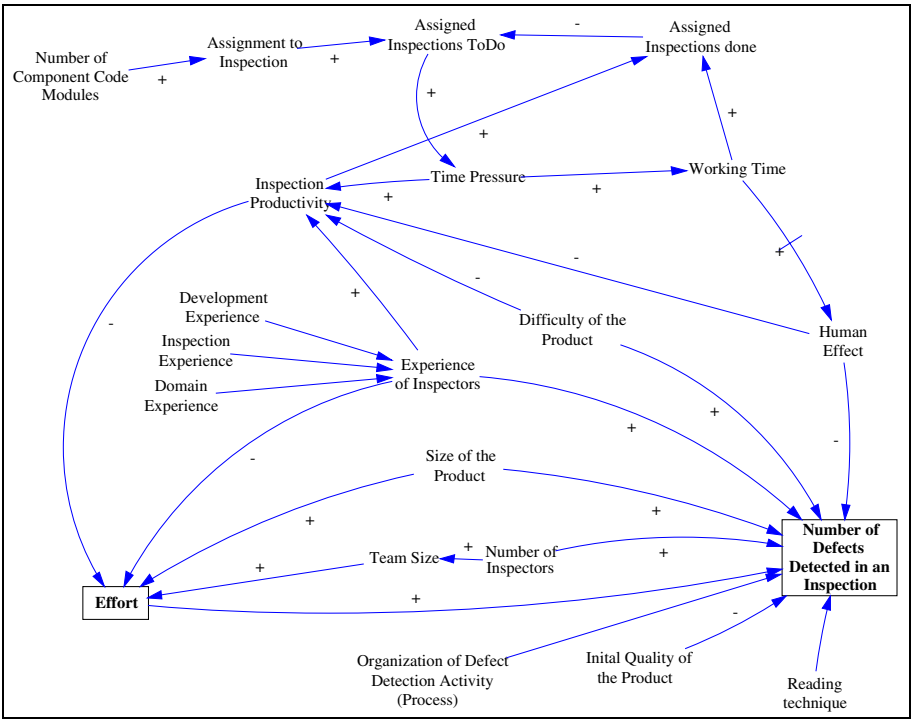

Fig. 4. Cause effect diagram for number of defects and effort

For inspection processes several models have been built with system dynamics; an elaborated model is described in [Mad96] and [Mad94]. In this model the relevant variables are the tasks, defects, and effort but there is no distinction beween the different tasks to do (coding or designing parts of the product and inspecting it). Furthermore, the defects are not connected to these different tasks or units in the model. The effort is determined by a pool of persons with average attributes.

With this system dynamics model some questions that can aid learning can not be answered. Examples questions are "Which documents should be inspected?" or "Should the tasks to be performed distributed unevenly among the available participants?"

The profile of an team can be modeled with a discrete event model, and what happens if the profile changes. Besides the individual modelling of the participants the other items e.g. code items can have attributes that characterize the single item and also provides a profile for the whole project. The size of a code item would be an interesting attributte because it can be used to determine the time for coding or the defects expected after coding.

These questions can be addressed with a discrete event model where also distributions can capture the uncertainity of assumptions like the number of defects in one code item.

If a discrete event model should be created the "moving units" (MU) have to be identified and their attributes that capture the information about the single MU in the model. A MU can model persons that perform different roles in the process and have different properties. These persons are connected with other MU that represent code items to be implemented. According to the individual properties of the person and the code item and a stochastic factor the coding time and the produced defects are computed. The same can be done with the inspection preparation and inspection meeting for the time needed and the defects found. If sufficient empirical data exists it is possible to use complex relations determined with decision trees or neural networks. Otherwise the relations have to be created from expert knowledge.

For building the simulation model the modeler has made assumptions based upon beliefs or analysis of empirical data of the process. In doing so, the modeler has gained a better understanding of the process, assuming that the simulation model has a behavior similar to the modeled process. If the simulation model has the expected behavior the simulation model can be used to conduct experiments. The results of the experiments with the simulation model help making decisions that could be made only with long years of experience and learning. For example, the introduction of inspections is sometimes difficult to motivate because of the additional effort that has to be spent. This is one reason why inspections are only used in few organizations. To decrease these doubts a general simulation model can be used and give the project managers more insight into the dynamics of a software process. For more precise results the model has to be fitted with company specific data and, depending on the process, changes have to be made to adopt the model to the company specific process.

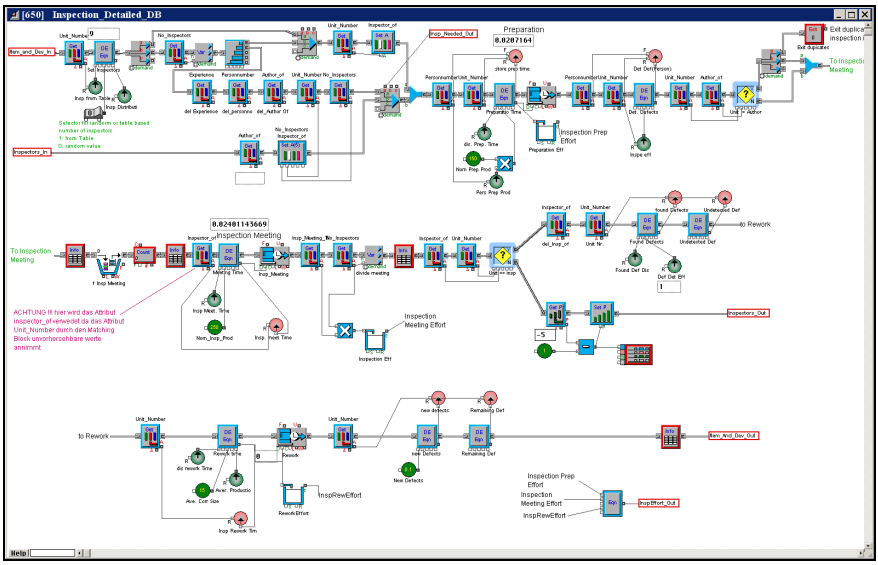

Fig. 5. Discrete event model of an inspection process

If a model is detailed an accurate enough, possible changes to the process can be tested that are too expensive and time consuming or even to dangerous and impossible to implement at the moment. Here the simulation helps to build a better understanding of the dynamics of the process and can aid decisions.

5 Related Work

Derek Merill and James S. Collofello [MeC97] present a Software Project Simulator that was used in the training for graduate students. They motivate the usage of the simulator with the need to gain enough experience that usually cannot be gained with reading or lecturing alone. Like pilots train in a simulator the students or projects managers can train with a simulator. The simulator is based on a system dynamics model and was created by [Tve96].

Pfahl und Ruhe [PfR01] address the process simulation in software organizations and how the simulation can be used to facilitate the learning and also the usage in combination with an experience factory. They studied System Dynamics in the context of organizational learning and took the areas of understanding, planning, controlling, and improving into account. They emphasize that learning based on system dynamics is an add on to the empirical learning supported by the experience factory approach.

Teved and Collofello [TvC95] state as motivation for their research the reduction of the cycle time and that the reduction was achieved in the most cases by changing the process and applying the CMM. The models can aid here to improve the evaluation of the effectiveness of purposed changes regarding the cycle time of the development process.

Pfahl, Klemm and Ruhe [PKR01] developed a CBT (Computer Based Training) module with a simulation component for the education and training in project management. They validated the simulator in an experiment with students. The simulator addresses the overall processes with requirement, design, coding and test. Optional quality activities (inspection) can be chosen. They state that the learning with a project simulator is as effective as providing the information with a detailed COCOMO model but also reason that the number of participants was to small to lead to significant statistical results.

6 Discussion

The normal way of learning includes feedback cycles: Process Performers or the organization change parts of the process and analyze what happens. If an experienced person is available, this person can act a teacher or mentor to explain effects and expected outcome of the change. Otherwise the change in the process or any other change that influences the organization has to be implemented explicitly and the outcome has to be analyzed. A successful change meeting the expected goals is a positive feedback loop and vice versa. To decide if an implemented change is actually an improvement, the changed process has be lived for a while and has to be measured in order to get the data for analyzing the change impact. This leads to feedback cycles that have a long delay time.

With the simulation of the proposed changes the process engineers, project managers or any other persons can optimize the feedback cycles. Several alternatives (if available) can be tested or - if a change has to be made because of other reasons - the outcome can be foreseen. When using simulation models, an important issue to consider is that positive results for a proposed change in a simulation model are no guarantee that the same change in the real process has a similar result. The model may lack one or more important relations that had to be considered to give a more precise result. A simulation model, however, can be used to detect unwanted effects: If a change does not have the expected outcome, especially a worsening effect, the simulation model falsifies the assumption in the context of the simulation model. Thus, simulation models are a means to avoid negative effects. Unwanted effects in the simulation of the model are clear indicators to be very careful when implementing this change in the actual process. Simulation models yield and approximation of the expected outcome regarding the changes, but cannot give exact results regarding the result parameters (e.g., duration, effort, quality).

A very important issue to consider in this context is the quality of the model; models that have been used for a longer time in the organization and are tested in several situations are more trustworthy than newly created ones. Here we can fall back on the works of Balci or Sargent among others, who work in the area of validation, verification of simulation models in general. The principles described there, for example in [Bal95], are also applicable to simulation models in the software engineering domain.

Developing software is a creative human-based task. Therefore the human factors should be included in the Models in order to gain most from the models. People learn during the execution of a project and their skills improve. If necessary, a distinction

between different skills is possible. Typical skills in software development could be coding productivity skill or a coding quality skill. But the participants suffer also fatigue effects as a result of continuous time pressure, which leads in the beginning to a rising performance and later on to a falling performance. With system dynamics models the human factors are easier to model because of continuous level variables and feedback loops. On the other hand individual humans are difficult to model, only the average or a few groups of different people can be modeled. For discrete event models each staff item has some skill attributes that could increase with a logistic growth each time an activity ends depending on the type and duration of the activity. Each staff item can start with individual skill values and if necessary with an individual learning factor provided from a database.

Despite the advantages of process simulation over running experimental processes, obtaining results from process simulation models requires a considerable effort. Thus, it needs to be carefully considered when to use process simulation. A way to reduce this effort could be a modular approach, which takes only into account particular aspects or subactivities in the process and create models for these aspects or subactivities. These parts of the model can be used in other models, which contain the same or similar aspects or subactivities. With these parts the creation of the model can be easier and faster because typical elements of a software development process can be reused and only be fitted to the specific problem. Examples are the coding activity followed by an Inspection or the human effects inherent in software development processes.

7 Summary and Outlook

This paper presented the simulation of descriptive software process models as a way to increase understanding of the dynamics of software development and presented an overall approach to develop simulation models. In the usage of simulation models, several issues remain open, which should be addressed by future research in order to make process simulation more feasible.

Developing a detailed simulation model takes time and effort. With an explicit method that gives guidelines for developing a discrete event model in the Software engineering domain the initial development of a simulation model could be easier and more efficient. Here, a modular approach that does not consider the entire model, but those aspects, which are particularly important for typical applications of simulation modeling, can help to define parts that can be used in other models.

With these typical clusters or micro models identified and described, their validity and usage could be described in a pattern-like approach. With such a set of model pattern the creation of simulation models could be faster and the validation of the model easier.

If a set of simulation patterns exist, simulation models could be created faster by reusing these patterns. Each pattern can consist a typical implementation of itself to be used if possible and speed the development of the simulation model further up.

The ideas for combining process modeling and process simulation for the purpose of learning in the context of a software organization was connected with the work in

the SEV Project (simulation based evaluation and improvement of software development processes) and research conducted at Fraunhofer IESE in the area of knowledge elicitation for descriptive process modeling.

In the ProSim project we will further investigate the possibilities for creating model patterns for faster model development.

Acknowledgements

The German ministry for education and research funds the SEV project; ProSim is funded by the foundation innovation of Rhineland-Palatinate.

References

[AK94] James W. Armitage and Marc I. Kellner. A Conceptual Schema for Process Definitions and Models. In Dewayne E. Perry, editor, Proceedings of the Third International Conference on the Software Process, pages 153–165. IEEE Computer Society Press, October 1994. [BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, Encyclopedia of Software Engineering, volume 1, pages 528–532. John Wiley & Sons, 1994.

[Bal95] Osman Balci, Principles and Techniques of Simulation Validation, Verification, and Testing, Proceedings of the 1995 Winter Simulation Conference; ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman; 1995, page 147- 154

[Bec01] Ulrike Becker-Kornstaedt. Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling. In Frank Bomarius and Seija Komi-Sirviö, editors, Proceedings of the Third International Conference on Product–Focused Software Processes Improvement (PROFES), Lecture Notes in Computer Science 2188, pages 312–325, Kaiserslautern, September 2001. Springer.

[CG98] Gianpaolo Cugola and Carlo Ghezzi. Software Process: a Retrospective and a Path to the Future. Software Process–Improvement and Practice, 4(3):101–123, September 1998.

[BHK+99] Ulrike Becker-Kornstaedt, Dirk Hamann, Ralf Kempkens, Peter Rösch, Martin Verlage, Richard Webby, and Jörg Zettel. Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance. In Matthias Jarke and Andreas Oberweis, editors, Proceedings of the Eleventh Conference on Advanced Information Systems Engineering (CAISE '99), pages 119–133, Heidelberg, Germany, June 1999. Lecture Notes in Computer Science 1626, Springer–Verlag.

[ChS00] Alan M. Christie and Mary Jo Staley, Organizational and Social Simulation of a Software Requirements Development Process, Software Process Improvement and Practice, 2000; 5 Pages: 103-110.

[Dow93] Mark Dowson. Software Process Themes and Issues. In Leon Osterweil, editor, Proceedings of the Second International Conference on the Software Process, pages 54–62, Berlin, Germany, February 1993. IEEE Computer Society Press.

[EbS93] Robert G. Ebenau, Susan H. Strauss, Software Inspection Process, McGraw-Hill, Inc. 1993.

[FFHM00] Raimund L. Feldmann, Michael Frey, Marco Habetz, and Manoel Mendonca. Applying Roles in Reuse Repositories. In Proceedings of the Twelfth Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, July 2000. Knowledge Systems Institute, Skokie, Illinois, USA.

[KBR+98] Marc I. Kellner, Ulrike Becker-Kornstaedt, William E. Riddle, Jennifer Tomal, and Martin Verlage. Process Guides: Effective Guidance for Process Participants. In Proceedings of the Fifth International Conference on the Software Process, pages 11–25, Chicago, IL, USA, June 1998. ISPA Press.

[Mad94] Raymond J. Madacy, A Software process dynamics model for process cost, schedule and risk assessment, PhD Dissertation, Department of Industrial and Systems Engineering, USC, December, 1994

[Mad96] Raymond J. Madachy, System Dynamics Modeling of an Inspection Based Process, Proceedings of the 18th International Conference on Software Engineering, 1996

[MaR00] Robert H. Martin, David Raffo, 2000, A Model of the Software Development Process Using Both Continuous and Discrete Models, Software Process Improvement and Practice, 2000; 5 Pages: 147-157

[MeC97] Derek Merrill, James S. Collofello, Improving Software Project Management Skills Using a Software Project Simulator, Frontiers in Education Conference, 1997.

[Pfa01] Dietmar Pfahl, An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations, Dissertation University of Kaiserslautern, 2001, Stuttgart: Fraunhofer IRB Verlag, 2001, xx, 284 S. Ill., Lit., ISBN: 3-8167-6066-X.

[PfL99] Dietmar Pfahl and Karl Lebsanft, Integration of system dynamics modelling with descriptive process modelling and goal-oriented measurement, Journal of Systems and Software, Volume 46, Issues 2-3, 15 April 1999, Pages 135-150.

[PfR01] Dietmar Pfahl, Günther Ruhe, System Dynamics as an Enabling Technology for Learning in Software Oranisations, Fraunhofer Institute for Experimental Software Engineering (IESE), IESE Report 025.01/E.

[PKR01] Dietmar Pfahl, Marco Klemm and Günther Ruhe, A CBT module with integrated simulation component for software project management education and training, Journal of Systems and Software, Volume 59, Issue 3, 15 December 2001, Pages 283-298.

[PWCC95] Marc C. Paulk, Charles V. Weber, Bill Curtis, and Mary B. Chrissis. The Capability Maturity Model for Software: Guidelines for Improving the Software Process. SEI Series in Software Engineering. Addison-Wesley, 1995.

[Ste00] John D. Sterman, Business Dynamics, Systems Thinking and Modeling for a Complex World, McGraw – Hill, 2000.

[TvC95] John D. Tvedt, James S Collofello, Evaluating the Effectiveness of Process Improvements on Software Development Cycle Time via System Dynamics Modeling, Proceedings of the 19th International Computer Software and Application Conferenc COMPSAC 1995.

[Tve96] J. Tvedt, An Extensible Model for Evaluating the Impact of Process Improvements on Software Development Cycle Time, PhD Dissertation, Arizona State University, Tempe, Arizona, 1996

[WB97] Richard Webby and Ulrike Becker. Towards a Logical Schema Integrating Software Process Modeling and Software Measurement. In Rachel Harrison, editor, Proceedings of the Nineteenth International Conference on Software Engineering Workshop: Process Modelling and Empirical Studies of Software Evaluation, pages 84–88, Boston, USA, May 1997.

Technology Support for Knowledge Management

Mikael Lindvall[1], Ioana Rus[1], and Sachin Suman Sinha[2]

[1] Fraunhofer Center for Experimental Software Engineering, College Park, MD20742, USA
{mikli,irus}@fc-md.umd.edu
[2] Dept. of Computer Science, University of Maryland at College Park, MD20742, USA
sachin@cs.umd.edu

Abstract. Human capital is the main asset of software organizations. Knowledge has to be preserved and leveraged from individuals to the organization. Thus, the learning software organization will be able to continually learn and improve. Knowledge management has various components and multiple aspects such as socio-cultural, organizational, and technological. In this paper we address the technological aspect; specifically, we survey the tools available to support different knowledge management activities. We categorize these tools into classes, based on their capabilities and functionality, and show what tasks and knowledge processing operations they support.

1 Introduction

Software organizations have realized that a large amount of problems are attributed to un-captured and un-shared product and process knowledge. Specifically they have discovered the need of knowing 'who knows what', the need for distance collaboration, and the need for lessons learned and best practices. This has led to a growing call for knowledge management (KM). Knowledge sharing and organizational learning can happen ad-hoc. It is, however, more efficient if organized. The amount of information and knowledge that needs to be captured, stored and shared, the geographic distribution of sources and consumers, and the dynamic evolution of information makes the use of software tools not an option, but a necessity. In this paper we characterize and classify software tools that support KM using two models, 1) knowledge characteristics (the *knowledge sharing* model) and 2) the evolution of knowledge (the *knowledge life cycle* model).

2 Knowledge Sharing and Knowledge Life Cycle Model

The knowledge-sharing model is also called the 'tacit-explicit model' (Nonaka and Takeuchi, 1995). Tacit knowledge is knowledge that rests with the employee and explicit knowledge is espoused knowledge. Conversion of knowledge from one form to another occurs frequently and often leads to the creation of new knowledge.

Explicit-explicit knowledge conversion or 'Combination' is the reconfiguration of explicit knowledge through sorting, adding, combining and categorizing. *Explicit-tacit* knowledge conversion or 'Internalization' takes place when one assimilates knowledge acquired from knowledge items. *Tacit-explicit* knowledge conversion or

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 94–103, 2003.

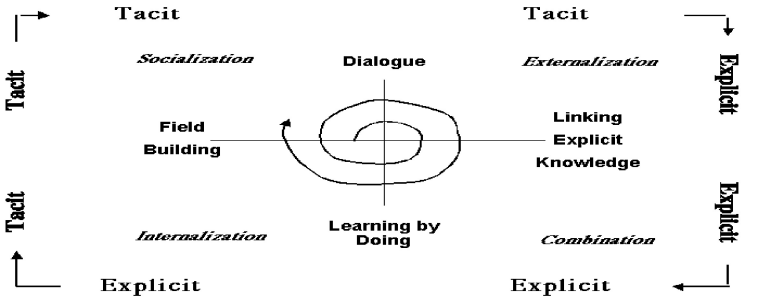

Fig. 1. The Tacit-Explicit Model (Nonaka and Takeuchi, 1995)

'Externalization' involves transforming quiet personal knowledge to espoused knowledge that can be either recorded or unrecorded. *Tacit-tacit* knowledge conversion or 'Socialization' occurs by sharing experiences, working together on a team, and by direct exchange of knowledge. A knowledge management system should support all four types of knowledge conversion.

The knowledge lifecycle (Wiig, 1999) takes the path of knowledge creation/acquisition, organization/storage, distribution, application/reuse and finally concludes with creation/acquisition again. Tacit knowledge has to be made explicit in order to be captured. This is accomplished with the aid of *knowledge acquisition* tools or *knowledge creation* tools. Knowledge acquisition evolves and builds an organization's knowledge base. Knowledge *organization/storage* refers to activities by which knowledge is organized, classified and stored in repositories. Explicit knowledge needs to be organized and indexed for easy browsing and searching. It must be stored efficiently to minimize storage space. A variety of tools have been developed to *distribute* or *deploy* knowledge. Knowledge can be distributed through various channels such as training programs, automatic knowledge distribution systems and knowledge-based expert systems. *Knowledge application* is the process through which knowledge becomes the basis for further learning and innovation. Applying knowledge is the payoff for knowledge management. A knowledge management system should support the entire knowledge flow in the knowledge life cycle.

3 Software Tools for Knowledge Management

KM should be supported by a collection of technologies for authoring, indexing, classifying, storing, and retrieving information, as well as for collaboration and application of knowledge. A friendly front-end and a robust back-end are the basic necessities of a software tool for knowledge management. Figure 2 shows a layered knowledge management system architecture based on (Lawton, 2001).

The lowest layer handles sources of explicit knowledge. Explicit knowledge resides in repositories as documents or other types of knowledge items (e.g., e-mail messages, and database records). Standard authoring tools (such as word processors) and database management systems (DBMS) support this layer. File servers, e-mail browsers etc. support the infrastructure layer. Document and content management tools with features for search and retrieval as well as analysis and maintenance repre-

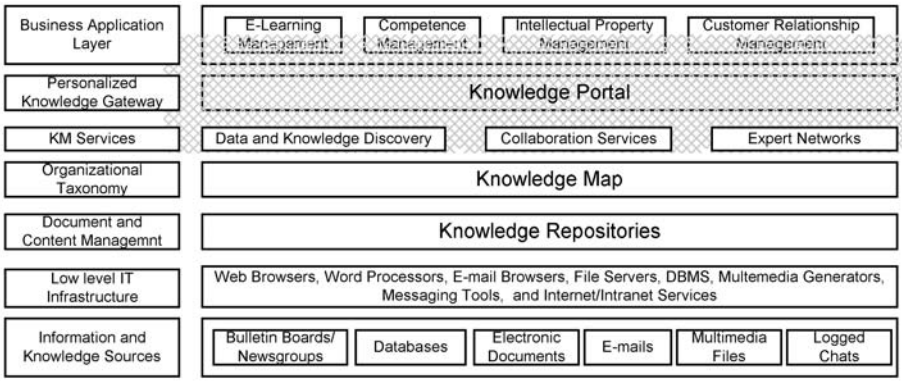

Fig. 2. KM Architecture Model

sent knowledge repositories. The organization of knowledge is based on a corporate taxonomy and serves as a "knowledge map" supported by classifying and indexing tools. KM services are provided using tools for data and knowledge discovery and collaboration services. Through portals, knowledge can be distributed to different users and applications, such as e-learning, competence management, intellectual property management, and customer relationship management.

It is hard to draw the line between information technology (IT) and tools for KM. In the architectural model above, we consider KM to be the higher layers, starting at the "knowledge repository" level, and IT constituting the lower layers, but this boundary is fuzzy. We concentrate on the upper layers of the architecture and analyze tools from a knowledge management perspective and discuss the needs of employees that use knowledge, the types of knowledge conversion that occur and a common set of features provided by tools from each category. The kinds of tools we analyze are those that are widely used for supporting the KM layers of the above architecture. This survey is not intended to cover all kinds of tools on the market, but to give the reader a sense of the various tools that are available for knowledge management.

3.1 Document and Content Management

In terms of knowledge management, the documents produced by organizations represent their explicit documented knowledge. New knowledge can be generated from documents; for example, de facto experts can be identified based on authors of documents. Document management systems enable explicit-to-explicit knowledge conversion. One could argue that a form of tacit-to-explicit knowledge conversion occurs when experts are determined based on the documents they authored. Expert identification certainly results in knowledge creation.

Common needs that arise in a document-sharing environment are related to identifying the latest version of a document, accessing documents remotely and sharing the documents in workgroups. Document management systems offer features that include storing/uploading of documents/files; version control; organization of documents in different ways; search and retrieval based on indexing techniques and advanced

searching mechanisms; and access from any Internet-connected workstation. Most document management systems also provide some kind of search for experts based on authorship.

Document management systems can aid learning software organizations that need to capture and share process and product knowledge. Most artifacts guiding a software project and being developed during a software project can be represented as documents and are the main explicit assets of the software organization. Document management systems thus help support the core business by managing these invaluable assets and enable transferring knowledge from experts to novices.

3.2 Collaboration Services

Delivering the right information at the right time is one of the major objectives of knowledge management. To achieve this, employees need to collaborate and communicate, especially when people work in an environment that is distributed in time and space. The knowledge conversions supported by this category of tools are mainly tacit-to-tacit, which occur, for example, when two or more users communicate using a chat tool or an instant messenger. One can argue that since the conversation is in an electronic form, a form of tacit-to-explicit conversion also takes place. Some tools make an effort to capture this conversation so that it can be published and used for other users. Some even analyze the conversation in order to create new knowledge, which strengthens the argument for a tacit-to-explicit knowledge conversion. The basic functionality of tools in this category is to connect employees by providing a computer-based communication channel. This communication can be synchronous or asynchronous. Collaboration in terms of chatting or white-boarding using a chat tool or a messenger tool would be an example of the former, while e-mail, bulletin boards, and newsgroups would be examples of the latter. A common practice is to use a tool to share a document in real-time so that two or more geographically distributed groups can see and hear the same presentation. Some tools are designed to capture communication and work results for further use and refinement. Some tools are designed to support concurrent co-authoring of documents over a distance. Other tools support active knowledge sharing in the form of e-learning. Some tools support eWorkshops in on-line moderated meetings between expert participants. The results of the workshop are captured and analyzed in order to generate new knowledge in a particular area. This illustrates that technology and process can be used to bring people together and generate new knowledge.

Distance collaboration is the need of the hour for learning software organizations. Due to globalization, software development working groups are often spread out geographically and work in different time zones. The outsourcing of subsystems to subcontractors also results in these geographically distributed teams working together. Collaboration tools will help these groups to communicate, collaborate, and coordinate independent of time and place.

3.3 Data and Knowledge Discovery

The goal of this category of tools is to generate new knowledge from existing data, information, and knowledge bases. Examples of tools include visualization and data

mining, as well as analysis and synthesis tools. Data mining tools try to reveal patterns and relationships between data and generate new knowledge about the underlying data and what it represents. Knowledge management tools often deal with raw data and singular data points, yet in order to create theories, these data points need to be analyzed. Raw data and singular data points are, for example, documents, frequently asked questions, lessons learned, and other knowledge items stored in knowledge bases. Data and knowledge discovery can reveal what is hidden in the data. Such tools can be used to identify patterns in the content and usage of knowledge. Knowledge discovery also identifies groups of users active, inactive, and de-facto experts. Such tools can also be used to analyze knowledge bases in order to generate more complex knowledge items, for example when deriving best practices based on lessons learned and frequently asked questions. The knowledge conversion that takes place as a result of data and knowledge discovery is, first and foremost, explicit-to-explicit due to the fact that all knowledge in the knowledge base is already explicit and the discovery process creates new explicit knowledge based on it. One could also argue that a portion of the explicit-to-tacit knowledge conversion occurs in the process when the analyst looks at the data from different perspectives and gains a better understanding of it.

Typical features of the tools in this category provide for visualization of data in different ways. Features for statistical analysis are also common, along with different features for decision support. These features are sometimes based on artificial intelligence (AI) techniques that can help in the discovery process. Some tools *analyze* multimedia content and transcribe it into text, identify and rank the main concepts within it, and automatically personalize and deliver that information to those who need it. Delivery can occur across the Internet or by using other digital channels such as mobile phones, handheld computers, etc. Other tools are used for organizing and analyzing knowledge and experience bases. Taxonomy editors enable classification managers to develop taxonomies for the experience base. Some tools connect to existing knowledge and experience bases and visualize their content. Analysis tools can be used to analyze growth trends of knowledge and experience bases, and for other related analyses.

The discovery of best practices and lessons learned is very important for learning software organizations. Software engineering has been advocating this for a long time. Software development teams work on similar kinds of projects without realizing that results would have been achieved more easily if they followed a practice adopted by a previous project.

3.4 Expert Networks

Expert networks provide a forum for people who need to establish knowledge sharing focused on solving a problem. Expert networks are typically based on peer-to-peer support. These kinds of systems aid geographically distributed organizations. Expert networks often emphasize the importance of acknowledging that most knowledge cannot be made explicit and stored in a computer, but will reside in the brains of experts. Peer-to-peer support is tacit-to-tacit when experts use a chat tool to communicate, but it is tacit-to-explicit when solutions are stored for future use and reference. One could also argue that an explicit-to-tacit conversion occurs when someone finds a solution to a problem in the knowledge base of stored solutions and applies it to solve

their task. Common features are expertise brokerage and expert identification. Other features are communication and collaboration between people and the capture of questions and answers. These tools typically track and rate expertise, customer satisfaction, and rewards that are handed out to people who contribute to the success of the system. Expert networks aid learning software organizations. Software development is a human and knowledge intensive activity and most of the knowledge is tacit and will never become explicit. It will remain tacit because there is no time to make it explicit. In this scenario Expert Networks can provide a solution as they can reduce the time spent by software engineers in looking for specific domain knowledge.

3.5 Knowledge Portals

Knowledge workers use many different computer-based information sources (sales results, manufacturing activities, inventory levels, and customer orders). These information sources need to be integrated and accessed through a common, yet personalized, interface. Portals create a customized single gateway to a wide and heterogeneous collection of data, information, and knowledge. They also provide different kinds of personalization so that content is presented in a manner that suits the individual's role within the organization and reflects personal preferences. Both the organization and the user can control what information is made available and how it is displayed. Portals pull information together from different sources and display it in a coherent way, performing an explicit-to-explicit knowledge conversion. Portals support knowledge distribution, as well as organization of the display of information.

Employees make decisions every day, but not all of them are informed. When critical data is hard to find, or takes too long to locate, it will not be available when it is needed for making a decision, and making the best decisions requires current and relevant information, which is what portals deliver.

Like other organizations, software organizations can benefit from portals. A study found that people in software organizations spent 40% of their time in searching for different types of information related to their projects (Henninger, 1997). With portals providing a one-stop shop for all the information that an individual wants, the time spent looking for information can be significantly reduced.

3.6 Customer Relationship Management

A popular area of knowledge management is customer support. There are mainly two forms of customer support tools: tools that enable customers to help themselves (self-help) and tools that help customer support personnel (help-desk). In some cases, vendors even set up areas for customers to help each other, i.e., to share knowledge about products and services (peer-to-peer). Customer support personnel might lack appropriate knowledge and consistency to deliver 24/7/365 support. This can be offset by systems that assist them with knowledge and support process, continuously and consistently, while they gain appropriate knowledge through experience. There are many cases where a high repeatability in the support process can be leveraged by reusing answers to the most common questions. Over time, support personnel also acquire a vast amount of knowledge about the products and services the organization offers, as well as information about customers and their behavior. This knowledge is a resource

for the organization as a whole and should be captured and spread. The knowledge conversation that takes place in customer support is mainly tacit-to-tacit, but with customer support systems that use knowledge bases it is possible to turn the process first into tacit-to-explicit, and then explicit-to-explicit, conversion. When customers search for (and later apply) knowledge, one can argue that explicit-to-tacit knowledge conversion takes place.

Applications for customer support are often based on knowledge repositories and, therefore, support the entire knowledge life cycle. Systems that support help desks typically have features that automatically direct customer requests to representatives based on profiles of the customers, as well as on the expertise of the representative. Past customer behavior and connections to product catalogs are other factors that can assist in the helping process. Support for self-help is often provided out of a website. Knowledge bases typically provide an interface to capture new knowledge about the products, services, and their use so that new cases, new incidents, and new lessons learned can be captured and shared. On-line customer support often links the self-help with the help desk through live chat and software systems that are capable of answering questions.

Software organization can greatly benefit from customer support systems. Most big organizations have already employed customer support systems to track bug reporting and crash histories of software. This helps in maintenance, version update and customer support for the users of that software.

3.7 Competence Management

Along with document management (DM), we view competence management (CM) as one of the most basic KM activities. If DM deals with the organization's explicit knowledge assets, then CM handles its tacit knowledge. Organizations need to develop 'knowledge maps' in terms of where knowledge items are and who knows what. Once such a knowledge map is in place it can be used to identify appointed and de facto experts; to staff new projects based on skills and experience required; to identify knowledge gaps that indicate the need to hire new people; or to develop training programs. Knowledge maps can also be used to identify core competencies for organizational marketing. Tools that support competence management become necessary, especially for large organizations, where people do not know each other. Their necessity also becomes obvious in any distributed, decentralized, and mobile organization. The knowledge transformations supported by these tools are mostly 'explicit-to-explicit' because they are based on repositories in which information about knowledge possession is stored. One can argue that tacit-to-explicit knowledge conversion takes place when people create profile about their own knowledge. One can also argue that knowledge creation takes place when the CM system analyzes the stored information and generates knowledge maps showing who knows what, or what competence the organization has or does not have.

A typical feature is profiling. Profiles of employees, customers, subcontractors, vendors, partners, projects, and positions can be generated, which also leads to identification of and searches for experts. Some tools automatically create competence profiles by mining various sources of information. Profiling mechanisms extract terms and phrases from e-mail communications and documents produced or shared by individuals. Each user profile provides a detailed index of an individual's knowledge,

experience, and work focus. A set of profiles, therefore, represents a composite 'snapshot' of the expertise within an organization. Competence management addresses a very important problem of learning software organizations, 'the need of knowing who knows what'. Much knowledge can be recorded for other organizations, but nevertheless the assets of a software engineering organization are mainly its employees and their tacit knowledge. As a knowledge intensive industry, software organizations are heavily dependant on tacit knowledge, which is very mobile. Competence management can help build structures and frameworks for capturing key information that can help retain some knowledge when employees leave or become unavailable.

3.8 Intellectual Property Management

Knowledge management often includes management of intellectual property such as patents, copyrights, trademarks, and service marks. Organizations that own intellectual property need ways to automate workflow and support the management and analysis of inventions, patents and related matters. It often takes a long time to file and obtained approved rights to intellectual property, and organizations need support to track this process, more so for international organizations. Intellectual property regulations require owners of copyrights, trademarks, and service marks to pay legal fees at specific points in time; otherwise the rights can be lost. For licensing issues, it is also important to track licensees and royalties. Another aspect of intellectual property is the protection of digital content covered by copyright. Intellectual property management is mainly an explicit-to-explicit knowledge conversion. It is based on knowledge repositories and, thus, deals with all aspects of knowledge storage, organization and knowledge distribution in a controlled way.

Typical features include search for patents capabilities, support to file for patents, searchable knowledge bases with rules and regulations and support for legal help, as well as collections of forms and standard letters. Other related issues that these tools support are licensing of patents and tracking of licenses, as well as calculation of fees.

3.9 E-Learning Management Systems

Knowledge management aims to help people acquire new knowledge, as well as package and deliver existing knowledge through teaching. *E-learning* is a relatively new area that includes computer-based and on-line training tools. E-learning is appealing because it offers flexibility in time and space, as well as collaboration between students and tutors. Many of the collaboration and communication tools mentioned before can be used to support this activity. *E-teaching* supports tacit-to-explicit knowledge conversion in that the teacher's tacit knowledge is converted to explicit learning material. E-learning supports explicit-to-tacit knowledge transformation in that students learn and internalize the explicit material. Both e-teaching and e-learning support tacit-to-tacit knowledge sharing when the tutor and student communicate. E-teaching involves knowledge creation, knowledge distribution, storage, and organization, as well as knowledge application (when the students apply the newly-acquired knowledge to problems).

Common features for tools include reusable learning object libraries; adaptive web-based course delivery; component-based authoring, scheduling and reporting tools;

student evaluation and progress tracking; and building of skills inventories. E-learning systems often include collaboration tools and support for different types of content, i.e., video, audio, documents etc. Searching for and matching of tutorials with student needs and on-line facilitation of these tasks, are often supported. Software development is becoming a more complex domain to master due to the constant change and stream of new technologies. Many industries have similar problems, but the software industry is probably worse than other industries due to the fact that the pace of change is higher. Newly emerging technologies cannot be mastered overnight. E-Learning tools can reduce the time and difficulty in learning by a considerable margin, thus helping software organizations remain competitive.

4 Summary

This paper focused on software tools for knowledge management. We have surveyed the commercial market for such tools, divided them into functional categories and described them from different perspectives. We provided two models that we used to describe the tools. All knowledge cannot be made explicit and recorded; thus a knowledge management solution must address both tacit and explicit knowledge. All repository-based software systems support a majority of the phases in the knowledge life cycle, while few systems actually deal with the analysis and synthesis of new knowledge. We conclude that most of the available tools are specialized, and not complete systems. This is a result of the many facets of KM that need to be addressed through a KM system. Another finding is that not all tools that are labeled 'KM tools' are indeed KM tools; their vendors attempt to make them more attractive by attaching this buzzword to them. Impressions from KMWORLD2001 and other conferences indicate that the market is steadily growing. To implement an efficient KM system, organizations must identify their main problems, priorities, and strategy, and then select appropriate tools. Knowledge management relies heavily on technology, but it is important to realize that technology alone will never be the solution to knowledge management. There are socio-cultural and organizational components that need to be addressed in a KM system implementation to assure its acceptance and success.

References

Henninger, S., "Case-Base Knowledge Management Tools for Software Development," *Automated Software Engineering*, vol. 4, pp. 319-340, 1997.

Lawton, G., "Knowledge Management: Ready for Prime Time?," *IEEE Computer*, vol. 34, no. 2, pp. 12-14, 2001.

Nonaka, I. and Takeuchi, H., *The Knowledge Creating Company,* Oxford University Press, 1995.

Wiig, K., Comprehensive Knowledge Management - Working Paper, http://www.knowledgeresearch.com/downloads/compreh_km.pdf, Knowledge Research Institute, Inc., 1999.

Resources

Web Resources

Document and Content Management	http://www.microsoft.com/sharepoint http://docushare.xerox.com http://www.lotus.com/home.nsf/welcome/km http://www.hyperwave.com http://www.documentum.com
Collaboration Services	http://www.microsoft.com/windows/netmeeting http://www.lotus.com/home.nsf/welcome/sametime http://www.lotus.com/home.nsf/welcome/quickplace http://www.groupsystems.com http://fc-md.umd.edu/eworkshop
Data and knowledge Discovery	http://www.autonomy.com http://www.spotfire.com http://www.digimine.com http://fc-md.umd.edu/vqi/
Expert Networks	http://www.abuzz.com http://www.sopheon.com/
Knowledge Portals	http://www.plumtree.com http://www.OptimalView.com http://www.ascentialsoftware.com http://www.lotus.com/home.nsf/welcome/kstation
Customer Relationship Management	http://www.askit.com http://www.askmecorp.com http://www.remedy.com http://www2.xerox.com/go/xrx/knowledgest/knowledge_section.jsp?id=19238
Competence Management	http://web.skillscape.com http://www.tacit.com
Intellectual Property Management	http://www.patentcafe.com http://www.contentguard.com
E-learning Management Systems	http://www.hyperwave.com/e/products/els.html http://www.wisdomtools.com http://www.firstdoor.com http://www.tutor.com

Further Reading

Agresti, W., "Knowledge Management," *Advances in Computers*, vol. 53, pp. 171-283, 2000.

Basili, V. R., Tesoriero, R., Costa, P., Lindvall, M., Rus, I., Shull, F., and Zelkowitz, M. V. "Building an Experience Base for Software Engineering: A report on the first CeBASE eWork-shop", Bomarius, Frank and Komi-Sirviö, Seija, Springer, *In Proceedings of* Profes (Product Focused Software Process Improvement), pp. 110-125,2001.

Lindvall, M., Rus, I., Jammalamadaka, R., and Thakker, R., Software Tools for Knowledge Management, DACS State-of-the-Art-Report, 2001.

Rus, I. and Lindvall, M. Knowledge Management in Software Engineering. IEEE Software 19[3], 26-38. 2002.

Rus, I., Lindvall, M., and Sinha, S., Knowledge Management in Software Engineering, DACS State-of-the-Art-Report, 2001.

Software Engineering Decision Support – A New Paradigm for Learning Software Organizations

Günther Ruhe

University of Calgary
ruhe@ucalgary.ca
http://sern.ucalgary.ca/~ruhe/

Abstract. Software development and evolution is characterized by multiple objectives and constraints, by a huge amount of uncertainty, incomplete information and changing problem parameters. Success of software development very much depends on providing the right knowledge at the right time, at the right place, and for the appropriate person. Experience factory and organizational learning approaches are increasingly used to improve software development practices.

The paradigm of Software Engineering Decision Support (SEDS) goes beyond the concept of reusing models, knowledge or experience. For more focused problem domain, emphasis is on providing methodology for generation, evaluation, prioritization and selection of solution alternatives. Typically, modelling, measurement, empirical and simulation-type investigations are combined with intelligent methods of analysis and reasoning to predict the impact of decisions on future life-cycle performance.

This paper describes fundamental principles and expectations on SEDS. A comparison with knowledge management-based approaches is performed for the areas of requirements negotiation and COTS selection. The initial hypothesis on the expected benefits of SEDS are discussed for the two case study examples in the area of requirements negotiations.

1 Introduction

The need for further development of software engineering practices within companies adds to the demand for systematic knowledge and skill management in combination with active usage of this knowledge to support decision-making at all stages of the software lifecycle. With continuous technological change, globalization, business reorganizations, e-migration, etc. there is a continuous shortage of the right knowledge at the right place at the right time. Subsequently, strategic and operational decisions concerning products, processes, technologies or tools and other resources, are far from being mature. Reactive management is the rule, and pro-active analytical performance is more the exceptional case.

Experience factory and organizational learning approaches are increasingly used to improve software development practices [15], [18]. The main idea of experience based learning and improvements are to accumulate, structure, organize and provide any useful piece of information being reused in forthcoming problem situations [2]. Reuse of know-how is essentially supported by the case-based reasoning methodol-

S. Henninger and F. Maurer (Eds.): LSO 2002, LNCS 2640, pp. 104–113, 2003.

ogy [1]. However, software development and evolution typically is large in size, of huge complexity, with a large set of dynamically changing problem parameters. In this situation, reuse of experience alone is a useful, but non-sufficient approach to enable proactive decision analysis. Diversity of project and problem situations on the one hand, and costs and availability of knowledge and information organized in a non-trivial experience (or case) base on the other hand, are further arguments to qualify decision-making.

The idea of offering decision support always arises when decisions have to be made in complex, uncertain and/or dynamic environments. The process of software development and evolution is an ambitious undertaking. In software development and evolution, many decisions have to be made concerning processes, products, tools, methods and techniques. From a decision-making perspective, all these questions are confronted by different objectives and constraints, a huge number of variables under dynamically changing requirements, processes, actors, stakeholders, tools and techniques. Very often, this is combined with incomplete, fuzzy or inconsistent information about all the involved artefacts, as well as with difficulties regarding the decision space and environment [13], [14].

Typically, a concrete decision support system is focused on a relatively narrow problem domain. There are two kinds of existing contributions to Software Engineering Decision Support. Firstly, an explicitly mentioned effort to provide decision support in a focused area of the software life cycle. Examples are decision support for reliability planning [17] or decision support for conducting inspections [9]. Secondly, this encompasses research results that indirectly contribute to decision support, although not explicitly stated as such. Basically, most results from empirical software engineering, software measurement or software process simulation can be seen to belong to this category.

The main purpose of this paper is to position SEDS as both complementary and supplementary to experience factory or learning software organization approaches. The concrete relationship is problem and context dependent. The paper is subdivided into five parts. Following this introduction is a characterization of Software Engineering decision-making. Software Engineering decision support systems (SE-DSS) couple the intellectual resources of individuals and organizations with the capabilities of the computer to improve the quality of solutions. They are described in more detail in part 3. This is followed in part 4 by an analysis of the concrete examples for offering support for crucial decisions. One example concerns requirements selection. The other is related to support in software release planning. Finally, the summary and an outlook are presented in part 5.

2 Why Do We Need Support for Making Decisions in Software Engineering?

Software Engineering is defined as [21]:

1. the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, that is, the application of engineering in software, and
2. the study of approaches as in 1.

Both parts of this definition imply a large number of detailed questions and necessary decisions on how to do that, i.e., they are concerned with SEDS. The demand for decision support covers the complete life cycle. For the analysis, design, construction, testing and evolution phase, decision makers need support to describe, evaluate, sort, rank, select or reject candidate products, processes, resources, tools and technologies. Example decisions are related to:

- **Requirements:** Which functional and non-functional requirements should be chosen according to the given budget and time constraints [16]? How to assign different requirements to releases under the assumption of an incremental development paradigm [7]?
- **Architecture and design:** How should the selection between candidate architectures be made to ensure the best fit in terms of software reliability, performance, security, and usability [22]? How to integrate available components and COTS products into a system design [14]?
- **Adaptive and corrective maintenance:** Which components need improvement during the maintenance cycle because of changing contexts and requirements [24]? Which modules are potentially low qualified during the development phase and need special emphasis during maintenance [20]?
- **Project planning and control:** What should the reaction be to shortages on budget, time or available resources? Which trade-offs are acceptable to deliver the product earlier? How should deficits in staff be compensated? How do we respond to violations of quality constraints for intermediate products?
- **Verification and validation:** Which technique is most appropriate? Which artefacts should be investigated [9]? When to terminate testing or inspections? Is there any need for re-inspections [6]? How to integrate components for system testing [5]?

Decision-making is a well-established discipline with origins and close interactions with many other disciplines such as economics, operations research, game theory, probability theory, control theory, psychology, and cognitive science. The emphasis of decision support is to provide as much background as possible for actually making the decision. This is a very essential input for the actual decision-maker (typically, a completely different person).

Decision support has been successfully designed, developed and applied in many areas such as logistics, manufacturing, health care, forestry or agriculture. Why do we also need decision support in software engineering? Some of the major concerns we encountered for current real-world situations in software development and evolution are summarized below:

- Decision problems are often poorly understood and/or described.
- Decisions are done at the last moment and/or under time pressure.
- Decisions are not relying on empirically evaluated models, best knowledge and experience and a sound methodology.
- Decisions are made without considering the perspectives of all the involved stakeholders.

Decisions are not explained or made transparent to those involved. What are the expectations and requirements for systems - offering SEDS? We define a set of “idealized” requirements on support systems that combine the intellectual resources of

individuals and organizations with the capabilities of the computer to improve effectiveness, efficiency and transparency of decision-making:

(R1) **Knowledge, model and experience management** of the existing body of knowledge in the problem area (in the respective organization).

(R2) **Integration** into existing organizational information systems (e.g., ERP systems).

(R3) **Process orientation** of decision support, i.e., consider the process how decisions are made, and how they impact development and business processes.

(R4) **Process modeling and simulation component** to plan, describe, monitor, control and simulate ("what-if" analysis) the underlying processes and to track changes in its parameters and dependencies.

(R5) **Negotiation component** to evolutionary find and understand compromises.

(R6) **Presentation and explanation component** to present and explain generated knowledge and solution alternatives in various customized ways to increase transparency.

(R7) **Analysis and decision component** consisting of a portfolio of methods and techniques to evaluate and prioritize generated solution alternatives and to find trade-offs between the conflicting objectives and stakeholder interests.

(R8) **Intelligence component** to support knowledge retrieval, knowledge discovery and approximate reasoning.

(R9) **Group facilities** to support electronic communication, scheduling, document sharing, and access to expert opinions.

Depending on the concrete problem topic and the usage scenario of the DSS (on-line versus off-line support, individual versus group-based decision support, routine versus tactical versus strategic support), different aspects will become more important than others.

3 Software Engineering Decision Support Systems – Basic Architecture

Software Engineering Decision Support Systems (SE-DSS) can be seen as an extension and continuation of the Software Engineering experience factory and LSO approaches. In addition to collecting, retrieving and maintaining models, knowledge, and experience in the form of lessons learned, SE-DSS generates new insights from on-line investigations in a virtual (model-based) world, from offering facilities to better structure the problem as well as in ranking and selecting alternatives. For this purpose, sound modeling and knowledge management is combined with a variety of techniques of analysis, simulation, and decision-making.

While the learning software organization approach is mainly addressing the learning aspect from an organizational perspective, the emphasis of real world SE-DSS is typically on more focused aspects of the software engineering life-cycle, e.g., resource planning, COTS selection or requirements negotiation.

The underlying hypotheses of using Software Engineering Decision Support Systems are:

Hypothesis 1: Quality: SE-DSS enables making more effective decisions.

Hypothesis 2: Efficiency: SE-DSS enables - making more efficient solutions.

Hypothesis 3: Transparency: SE-DSS allows more transparent decisions (to be better understood by involved individuals), reflecting trade-offs between conflicting criteria or stakeholder opinions.

Hypothesis 4: Stability: SE-DSS can be used to propose more robust decisions (stable under slightly changing environments).

Hypothesis 5: Transparency: SE-DSS in combination with proper modelling, optimization and simulation facilities can be used to generate and evaluate new solution alternatives and to better react on changes in the problem parameters.

Ideally, a SE-DSS should have simulation facilities to conduct scenario-based experiments in a virtual world. Simulation models can be used to systematically develop and evaluate improvement suggestions in a virtual (laboratory-like) setting. Similar to systematic experiments in the real world, a simulation model can be used to investigate whether changes in model parameters or model structure improve model behaviour with respect to specified goals or thresholds. In order to do so, proposed changes of the real system are implemented in the model and then compared to the baseline behaviour. If several improvements are suggested, the one with the highest impact can be identified. In addition to that, the effect of combining several improvement suggestions can be analysed [11].

The principal architecture of a SE-DSS is shown in Figure 1. Real-world decisions in planning, development or evolution processes in Software Engineering are done by humans. All support is provided via a graphical user interface. Experts and their human intelligence is integrated via group support facilities. The intelligence of the support is based on a comprehensive model, knowledge and experience base. The more reliable and valid the models are, the more likely we can expect good support. The accompanying suite of components interacts with the model, knowledge and experience base. The suite encompasses tools for modeling, simulation, as well as decision analysis. Furthermore, intelligent components for reasoning, retrieval and navigation are added to increase efficiency and effectiveness of the support.

4 Decision Support in Requirements Negotiations

4.1 Decision Support for Requirements Negotiations

Defining, prioritising, and selecting requirements are problems of tremendous importance. In [16], a new approach called Quantitative WinWin for decision support in

requirements negotiation is studied. The difference to Boehm's [4] groupware-based negotiation support is the inclusion of quantitative methods as a backbone for better and more objective decisions. Like Boehm's original WinWin [3], Quantitative WinWin uses an iterative approach, with the aim to increase knowledge about the requirements at each iteration.

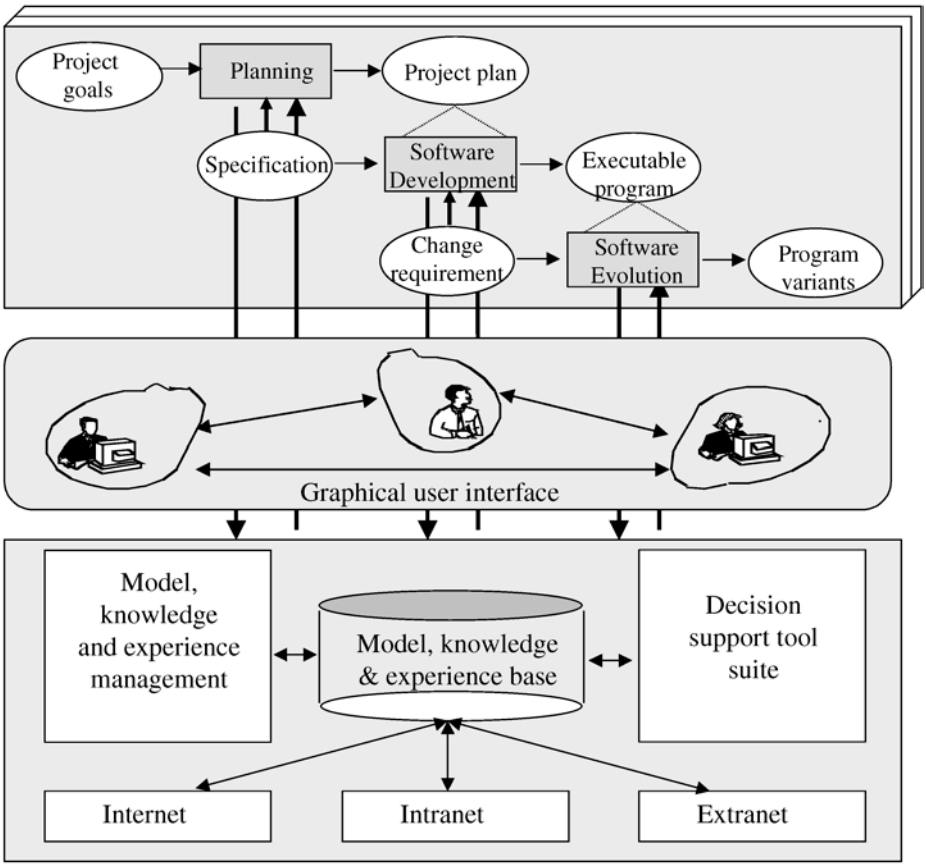

Fig. 1. Principal Architecture of a Software Engineering Decision Support System.

The overall method uses the Analytical Hierarchy Process [19] for a stepwise determination of the stakeholders' preferences in quantitative terms. These results are combined with methods for early effort estimation, in our case using the simulation prototype GENSIM [10], to evaluate the feasibility of alternative requirements subsets in terms of their related implementation efforts. As a result, quantitative WinWin offers decision support for selecting the most appropriate requirements based on the preferences of the stakeholders, the business value of requirements and a given maximum development effort.

How can the hypotheses formulated in chapter 3 work in this case? We don't have a quantitative evaluation yet, but we can briefly discuss the main arguments support-

ing the formulated hypotheses. The comparison is between using a DSS and subjective decision-making without any (quantitative) tool support:

Hypothesis 1: Improved quality: Quantitative Win-Win is an evolutionary approach taking into account all the information available at that moment to find the most appropriate subsets of requirements. The underlying algorithms are well established. The quality of the solutions provided mainly depends on the quality of the input data and the input model. However, as the used solution algorithms are objective, results should be better than those based on subjective selection of requirements. In addition to that, the system will provide a set of candidate solutions. Among them, the actual decision-maker can chose from.

Hypothesis 2: Efficiency: Effort to generate solutions related to their quality is improved under the assumption that models and relevant data are available. This needs an upfront investment, especially to create the effort estimation based on simulation runs.

Hypothesis 3: Transparency: From the application of Quantitative Win- Win you will get a preference structure among all the solutions generated. The preference is a result of systematic and pair-wise comparison between stakeholders and the requirements class alternatives. The final selection of requirements can be exactly linked to the chosen preference structure.

Hypothesis 4: Stability: Stability of the chosen solutions can easily be checked by computation of the stability intervals. This is relatively easy because of the power of the underlying algorithms (as opposed to subjective judgements to evaluate solutions).

Hypothesis 5: Flexibility: Quantitative Win-Win allows for following scenarios and varying problem parameters. Any changes in effort estimates, priorities or other problem parameters can be easily investigated with Quantitative WinWin. This also enables the generation of new solution alternatives.

4.2 Decision Support for Release Planning in Incremental Software Development

To achieve higher flexibility and to better satisfy actual customer requirements, there is an increasing tendency to develop and deliver software in an incremental fashion. In adopting this process, requirements are delivered in releases. Thus, a decision has to be made on which requirements should be delivered and in which release. Three main considerations that need to be taken into account are the technical precedence constraints inherent in the requirements, the typically conflicting priorities as determined by the representative stakeholders, as well as the balance between required and available effort. The technical precedence constraints relate to situations where one requirement cannot be implemented until another is completed or where one requirement is implemented in the same release as another. Similarly, certain requirements should be implemented in the same release. Stakeholder preferences may be based on the perceived utility or urgency of delivered requirements to the different stakeholders involved.

A method called EVOLVE is presented in [7] for optimally allocating requirements to increments. Methodologically, it is relies mainly on genetic algorithms, the principles of incremental and evolutionary software process models and aspects of greedy algorithms and the Analytic Hierarchy Process. EVOLVE typically generates a small set of most promising candidate solutions from which the actual decision-maker can choose.

We briefly discuss again the main contributions of EVOLVE in light of the above five hypotheses. The comparison is between using a DSS and subjective decision-making without any (quantitative) tool support:

Hypothesis 1: Improved quality: The proposed planning problem is highly complex (NP-complete) and cannot be expected to be solved adequately by individual judgement and trial and error type methods. Even Greedy-type heuristics are not competitive in terms of quality [7].

Hypothesis 2: Efficiency: Effort to generate solutions related to their quality is much lower than for any other method

Hypothesis 3: Transparency: The underlying fitness score function of EVOLVE guarantees optimal balancing between different stakeholder preferences, and this makes the proposed transparency.

Hypothesis 4: Stability: Stability of the chosen solutions can be judged from the different runs of the evolutionary algorithm (eventually, with varying crossover and mutation rates).

Hypothesis 5: Flexibility: EVOLVE allows investigation of any changes in requirements, priorities or other problem parameters. This enables the generation of new solution alternatives. Furthermore, variations of the weighting parameter in the objective function result in offering a set of most promising candidate solutions.

5 Summary and Conclusions

There are very good reasons for offering support for making decisions at the various stages of software development and evolution. Most of the related problems are very complex including different stakeholder perspectives and constraints. Mostly, decisions have to be made under uncertainty and incompleteness of information. Nevertheless, making good decisions is of tremendous importance for developing software faster, cheaper and of higher quality.

Currently, there is an increasing effort not only to measure or model certain aspects of the development processes, but to go further and integrate all available data, information, knowledge and experience with a sound methodology to provide the backbone for making good decisions. This mainly includes searching for all the objectives and constraints that influence a decision as well as elaborating on the defined solution space for possible courses of action. Typically, the different courses of action are noncomparable because of the different involved perspectives and objectives. This is exactly the borderline between offering decision support as addressed in the article,

and real-world decision making selecting among a range of alternative solutions generated by the intelligent pre-processing steps of SEDS.

As characterized by [23], decision support is most appropriate for semistructured and unstructured problems with emphasis on managerial control. What can be expected from decision support in the area of software engineering is higher decision quality; improved communication between all involved parties, increased productivity, time savings, and improved customer satisfaction. To achieve this goal, further effort should focus on (i) advancing SEDS methodology, especially by integrating aspects of decision-making under uncertainty, (ii) developing knowledge and experience-based software engineering decision support systems offering intelligent support on demand and via the web, (iii) further implementation and industrial evaluation of SEDS methodology and SE-DSS's, and (iv) evaluation of the underlying research hypotheses one to four describing the impact of SEDS on software development and evolution.

Acknowledgement

The author would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for the financial support of this research. Special thanks to the reviewers for their valuable comments that contributed to an improved version of the paper.

References

1. K.D. Althoff, "Case-Based Reasoning", in: Handbook of Software Engineering and Knowledge Engineering (SK Chang, ed), Vol. 1, pp 549-588.
2. V. Basili, G. Caldiera, D. Rombach, "Experience Factory". In: J. Marciniak: Encyclopedia of Software Engineering", Volume 1, 2001, pp 511-519.
3. B.W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, 21 (5), pp. 61-72, 1988.
4. B.W. Boehm, P. Grünbacher, B. Briggs, "Developing Groupware for Requirements Negotiation: Lessons Learned", IEEE Software, May/June 2001, pp. 46-55.
5. L.C. Briand, J. Feng, Y. Labiche, "Experimenting with Genetic Algorithm to Devise Optimal Integration Test Orders", Technical Report Department of Systems and Computer Engineering, Software Quality Engineering Laboratory Carleton University, 2002.
6. L.C. Briand, K. El-Emam, B. Freimut, O.Laitenberger, "A comprehensive evaluation of capture-recapture models for estimating software defect content", IEEE Transactions on Software Engineering, vol.26 (2000), pp 518-540.
7. D. Greer, G.Ruhe, "Software Release Planning: An Evolutionary and Iterative Approach", submitted to IST (2002).
8. H.W. Hamacher, G. Ruhe, "On Spanning Tree Problems with Multiple Objectives". Annals of Operations Research 52(1994), pp 209-230.
9. J. Miller, F. Macdonald, J. Ferguson, "ASSISTing Management Decisions in the Software Inspection Process", Information Technology and Management, vol.3 (2002), pp 67-83.
10. D. Pfahl.: "An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisation". Ph.D. thesis, University of Kaiserslautern, Department of Computer Science, October 2001.

11. D. Pfahl, G. Ruhe, "System Dynamics as an Enabling Technology for Learning in Software Organisations", 13th International Conference on Software Engineering and Knowledge Engineering. SEKE'2001, Skokie: Knowledge Systems Institute, 2001, pp 355-362.
12. S. Pfleeger, "Making Goog Decisions: Software Development and Maintenance Projects", Tutorial at 8th IEEE Symposium on Software Metrics, 2002.
13. G. Ruhe, "Software Engineering Decision Support: Methodology and Applications". Appears in: Innovations in Decision Support Systems (Ed. by Tonfoni and Jain), Springer 2003.
14. G. Ruhe, "Intelligent Support for Selection of COTS Products", appears in: Proceedings of the Net.ObjectDays 2002, Erfurt, Springer 2003.
15. G. Ruhe, "Learning Software Organisations". In: Handbook of Software Engineering and Knowledge Engineering (S.K. Chang, ed.), World Scientific Publishing 2001, Vol 1, pp 663-678.
16. G. Ruhe, A. Eberlein, D. Pfahl, "Quantitative WinWin - A New Method for Decision Support in Requirements Negotiation, Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'2002), pp 159-166
17. I. Rus, J.S. Collofello, "A Decision Support System for Software Reliability Engineering Strategy Selection", Proceedings of the 23rd Annual International Computer Software and Applications COMPSAC 99, Scottsdale, AZ, October 1999, pp 376-381.
18. I. Rus, M. Lindvall, "Knowledge Management in Software Engineering, IEEE Software May/June 2002, pp 26-38.
19. T.L. Saaty, "The Analytic Hierarchy Process", Wiley, New York, 1980.
20. N.F. Schneidewind, "Software Quality Control And Prediction Model for Maintenance", Annals of Software Engineering, vol. 9 (2000), pp 79-101.
21. SEWBOK. Guide to the Software Engineering Body of Knowledge. Version 0.95. IEEE Computer Society, May 2001.
22. M. Svahnberg, C. Wohlin, L. Lundberg, M. Mattsson, "Quality Attribute Driven Selection of Software Architecture Structures", Proceedings of the First Workshop on Software Engineering Decision Support, SEDECS'2002, Ischia, pp 819-826.
23. E. Turban, J.E. Aronson, "Decision Support Systems and Intelligent Systems", Prentice Hall, 2001.
24. G. Visaggio, (2000) "Valued-Based Decision Model For Renewal Processes in Software Maintenance", Annals of Software Engineering, vol.9 (2000), 215-233.

Author Index