PROCESS-ORIENTED KNOWLEDGE MANAGEMENT FOR LEARNING SOFTWARE ORGANIZATIONS

Frank Maurer[*] & Harald Holz[**]

[*]University of Calgary
Department of Computer Science
Calgary, Alberta, Canada, T2N 1N4
maurer@cpsc.ucalgary.ca

[**]University of Kaiserslautern
Department of Computer Science
D-67653 Kaiserslautern, Germany
holz@informatik.uni-kl.de

ABSTRACT

This paper discusses how a process-centered knowledge management and coordination support approach can be used to create learning software organizations. We discuss how process models can be used in project planning and how project plans can be enacted. Then we illustrate how a feedback loop can be created to update the process model stored in an experience factory. The result is a knowledge management approach that is process-oriented and supports continuous process improvement.

1. INTRODUCTION

Creating effective knowledge management structures is one of the key factors in software process improvement initiatives (like the Capability Maturity Model, Spice, Trillium, etc.). Most often the knowledge management needs are only mentioned implicitly. Specific organizational structures (e. g. a software process group) are developed for the purpose of managing and distributing knowledge about software development. These structures are costly to maintain and improving the efficiency of their members by a dedicated tool infrastructure would be very useful. This is especially true in case of virtual software corporations that collaborate using the Internet as their primary means of communication and information exchange.

The MILOS project of the University of Calgary and the University of Kaiserslautern aims at developing this infrastructure. Its primary goals are

- to provide access to relevant task-related knowledge when the task is executed
- to integrate knowledge management with project planning and process coordination
- to create a feedback loop from process execution to knowledge management resulting in support for learning software organizations
- to develop tool support (the MILOS system) for this closed-loop approach

Section 2 discusses our process-centered knowledge management approach. In section 3, we introduce the feedback loop for updating the process models. An example for our approach is given in Section 4. The last two sections discuss related work and evaluate our work.

2. PROCESS-CENTERED KNOWLEDGE MANAGEMENT

Our approach distinguishes between three kinds of knowledge:

- Generic, reusable knowledge
- Knowledge on specific projects
- Project data

These kinds of knowledge needed in software development projects are *structured in a process-oriented fashion*: Knowledge is linked to processes to be carried out in the course of the project. Processes are in the core of the knowledge organization and are our primary means for indexing information.

Generic Process Models are reusable descriptions of the workflow of software development processes. We associate generic knowledge to entities of the process model. Knowledge may be stored in several forms:

- Primarily, generic process models describe goals of tasks, ways how to solve tasks, conditions describing when a task can be started and which criteria its results must fulfill, required qualifications for tasks, the information flow between tasks.
- Context information pointing to background knowledge related to tasks
    - Concrete knowledge chunks to be interpreted by humans are stored as context information: URL references pointing to HTML pages or other files containing information in specific document formats. Examples are checklists for reviews or manuals for specific tools. Other examples would be document templates for specific tasks, e.g. a bug report template or a template for documenting test cases.
    - Predefined queries are used when the knowledge chunk can not be defined explicitly. The query describes the knowledge needed to perform a given task intentionally. We distinguish between several kinds of queries:

Experience base queries: We envision that our system will be connected to an experience base storing schedule, effort, cost, and quality information on past projects. This experience base stores structured information and could, e.g., be a relational database. To access this knowledge, the process model associates predefined queries with tasks[1]. When a user carries out the task, he will be able to execute the query. The system will connect to the database and present the results to the user. An example for such knowledge is a query that determines the average time spent for the coding of one class. Such a query could be linked to a task "Estimate implementation effort" and would help an inexperienced project manager to determine the overall time needed for implementing a system. Another example is a query that determines the testing effort for projects with a high meantime between failure.

Information retrieval requests: While the experience base stores structured information, most of the information produced or needed in software development is unstructured and informal or semi-formal at best. Requirements are typically documented as natural language text. User interface requirements often contain screenshots of mockups. System designs are represented with graphical notations (e.g. UML or ER diagrams) that often use textual annotations to make things clear. These representations can – partially – be accessed using standard information retrieval mechanisms. An IR request can be posted to one of the public web search engines or to an in-house document management system. An example is to find the newest version of the Java EJB package for a task "Implement Application Server" (the query would have to be posted to the Javasoft retrieval engine).

CBR requests: Case-based reasoning technology integrates structured and unstructured queries. It supports similarity-based queries on structured data whereas relational databases mainly use Boolean queries and IR systems work with unstructured data. If a company would annotate software life cycle documents with attributes and store them in a case base, then one could imagine to apply CBR technology for software process support. We mention CBR because it is often discussed in connection with KM for software processes. So far, we haven't seen a convincing argument why it is better suited for experience management than the technologies discussed above.

Project Plans tailor generic process models to the needs of a concrete project. This includes

- selecting processes and methods from the generic process model that should become part of the project plan. For example, the generic process model contains a process "testing" with methods "White box testing" and "Code reading". The project manager adds a "Testing" process to the project plan and then selects "code reading" as his method of choice.

- standard project planing: Assigning tasks to responsible agents, scheduling tasks, cost estimation etc. These task are commonly supported by COTS tools like MS Project.

Tailoring process models is a difficult task and many problems are still unresolved, e.g.:

- how to determine automatically if a generic model can be reused in a given situation

- how to support the customization process.

[1] In the following, task and process are used synonymously.

Currently, our approach does not support the tailoring process except by providing the planner with access to the generic process models. The decision which tasks to include in the plan and which methods should be used for solving a task stays with the planner.

Project plans contain the knowledge about tasks to be done and links to the knowledge contained in generic process models. They are a basis for project enactment and coordination. Using a process enactment engine in a project, a project plan is the basis for *actively guiding* human users in their work.

Project Data includes all information that will be created when tasks are executed – output products as well as additional information used in the task execution. During enactment, a team member is able to add information resources to a task. These could be URLs to sites in the intranet and/or Internet that provided valuable information for solving the task, additional queries to the experience base that he used to gather information, etc. This information is fed back to the generic process model during experience packaging (see Section 3).

Our flexible workflow/process engine handles all project data. It manages the state of the work process and its tasks, to-do lists for its users, the products created during process enactment, and traceability relationships between process entities (Dellen, Kohler, Maurer 1996). This information is created during process execution and, obviously, is highly relevant for the current project. In software development processes this includes e.g. requirements specifications, design documents, design rationales, traceability matrixes, source code etc.

Our approach provides - beside a product-oriented view coming from the software configuration management system used - also a *process-oriented view* on the data created during task enactment. Users are able to access information based on the processes carried out and they can follow the information flow in the project (thereby tracing where and by whom a specific information was used).

3. CREATING A FEEDBACK LOOP FOR CONTINUOUS LEARNING

To support an organizational learning process, our approach links process-centered knowledge management, with project planning, enactment support and experience packaging. We follow a four step process (see Fig 1).

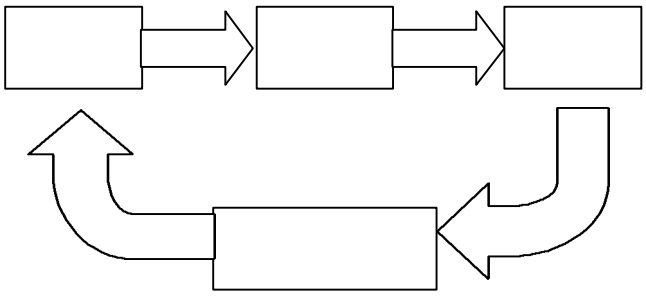

Figure 1: Knowledge Utilization Cycle

Process models describe generic tasks, their decomposition into subtasks, and the information flow between tasks (for a more detailed description see (Maurer, Dellen 1998)).

The library of process models contains descriptions of best practices in software development for a given company. In step 1, the project manager selects processes and methods from the library creating an initial project plan. This plan includes references to background knowledge (context URLs and queries) that were stored in the process model. The plan is uploaded to standard project management tools (currently we support MS-Project 98 and use it for scheduling and assigning tasks to resources). Changes carried out with MS-Project are then downloaded to our system. In the upload/download process, we preserve information flow dependencies between tasks as well as links to background knowledge coming from the process model.

The planning process is incremental: We can change the plan at any time during process execution. The current project plan is the basis for enactment supported by our workflow engine WFE (2). Using a standard Web browser, a user is able to connect to the WFE, access her to-do list, accept/reject/restart a task, access task inputs, and edit task outputs. After completing a task, the outputs are accessible as input for the successor tasks.

The next step is to extract reusable process knowledge from the current project plan (3). A user will be able to select parts of the project plan (e.g. a method for a specific task or the newly added information resources) and tell the system to upload this to the generic process model library.

The upload process will generate an equivalent partial model in the library that stores all process models (4). It has to make sure that the new information is properly integrated into the library and irrelevant information is deleted (e.g. there is no point in storing the concrete schedule of a specific project in a generic library).

4. THE MILOS SYSTEM

In the following, we want to illustrate our system's functionality with the help of an example scenario describing our own project's software development process.

Using the requirements document and the architecture description as a basis, the project planner browses the process model shown in Fig. 2. He retrieves the process type “Develop Component”. For each of the three architectural components *Project Plan Management* (PPM), *Workflow Engine* (WFE) and *Resource Pool* (RP) component to be developed during the project, he inserts an appropriately named instance of this type into the plan (e.g. “Develop WFE Component”).

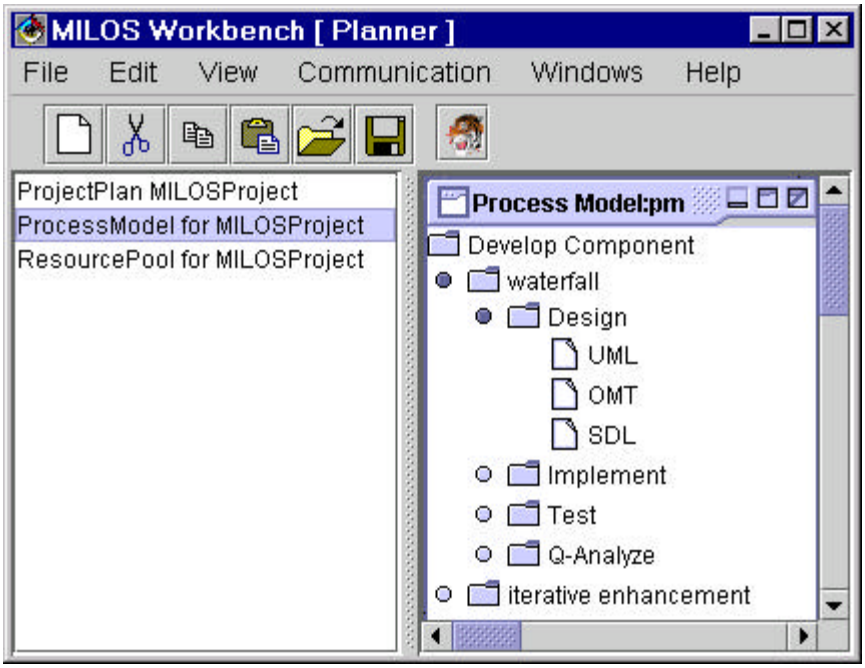

Figure 2: Example process model.

The process type “Develop Component” maintains knowledge about different development methods, e.g. “waterfall” or “iterative enhancement”. Hence, the planner can proceed by selecting one of these methods for the process “Develop WFE Component”(see Fig. 3). By choosing the method “waterfall”, instances of the method’s subprocess types “Design”, “Implement”, “Test” and “Q-Analyze” are automatically inserted into the plan. In addition to the knowledge chunks and queries these instances obtain from the model, they specify their required input documents and output documents to be produced during process execution. The planner is alerted to any input documents that are not part of the information flow defined by the method, and hence have to be produced by some other processes in the plan.

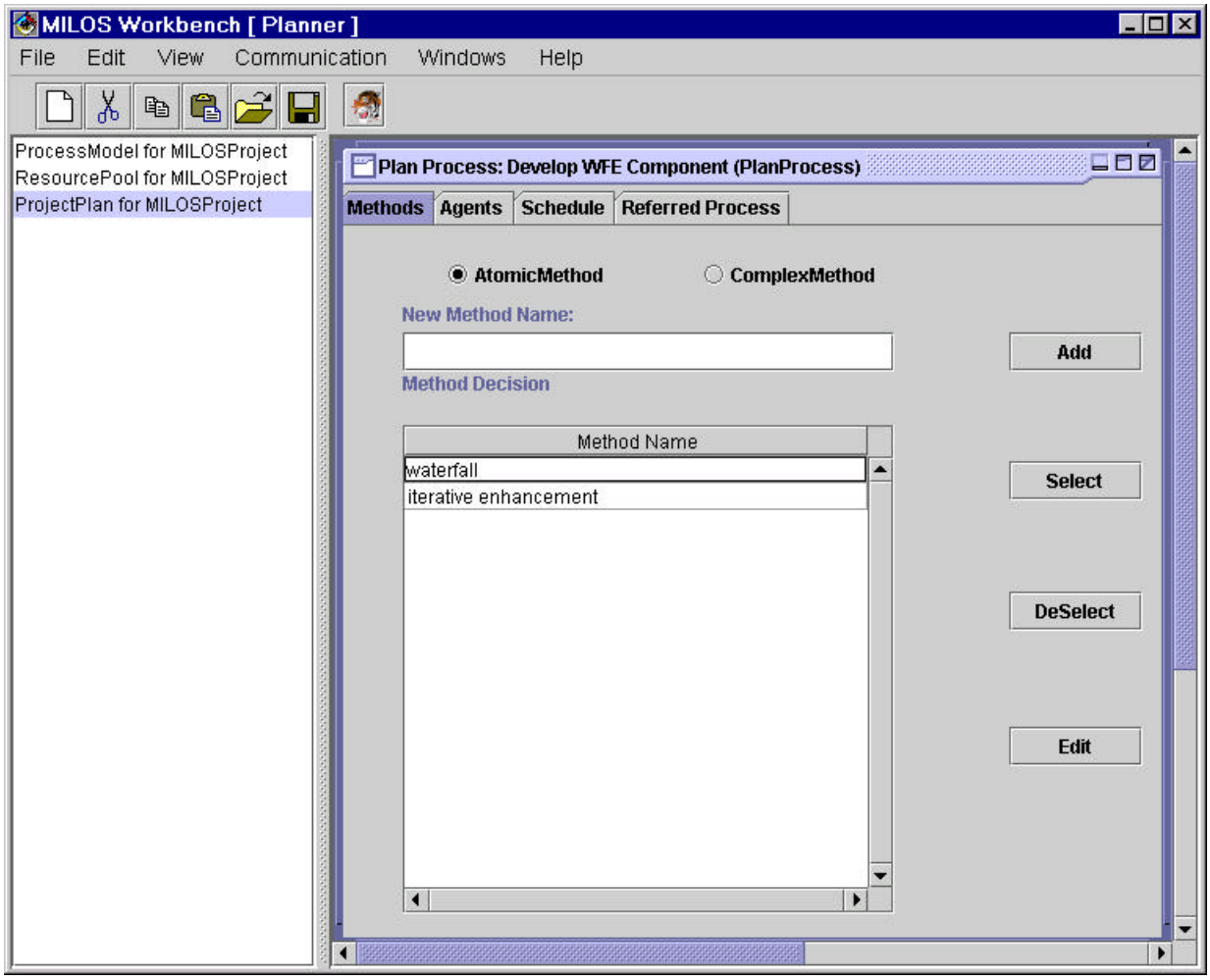

Figure 3: Planning decision.

Next, the planner decides on the design method to be used for the process “Design WFE”. The corresponding process type already contains specifications for the methods “UML”, “OMT” and “SDL”. After having settled on “UML”, he queries the project’s resource pool for qualified personnel that match the profile specified for this method in the process model. Since more than one team member has both design experience and familiarity with UML, the project planner tries to refine his query. He searches in the company’s project plan database for former instances of type “Design”. In particular, he looks for former design processes whose task description contains keywords like “workflow” or “event”, since the task at hand is the design of a flexible workflow engine based on event handling. Since only for Bob several design processes are found that dealt with event handling, the planner assigns the process “Design WFE” to him.

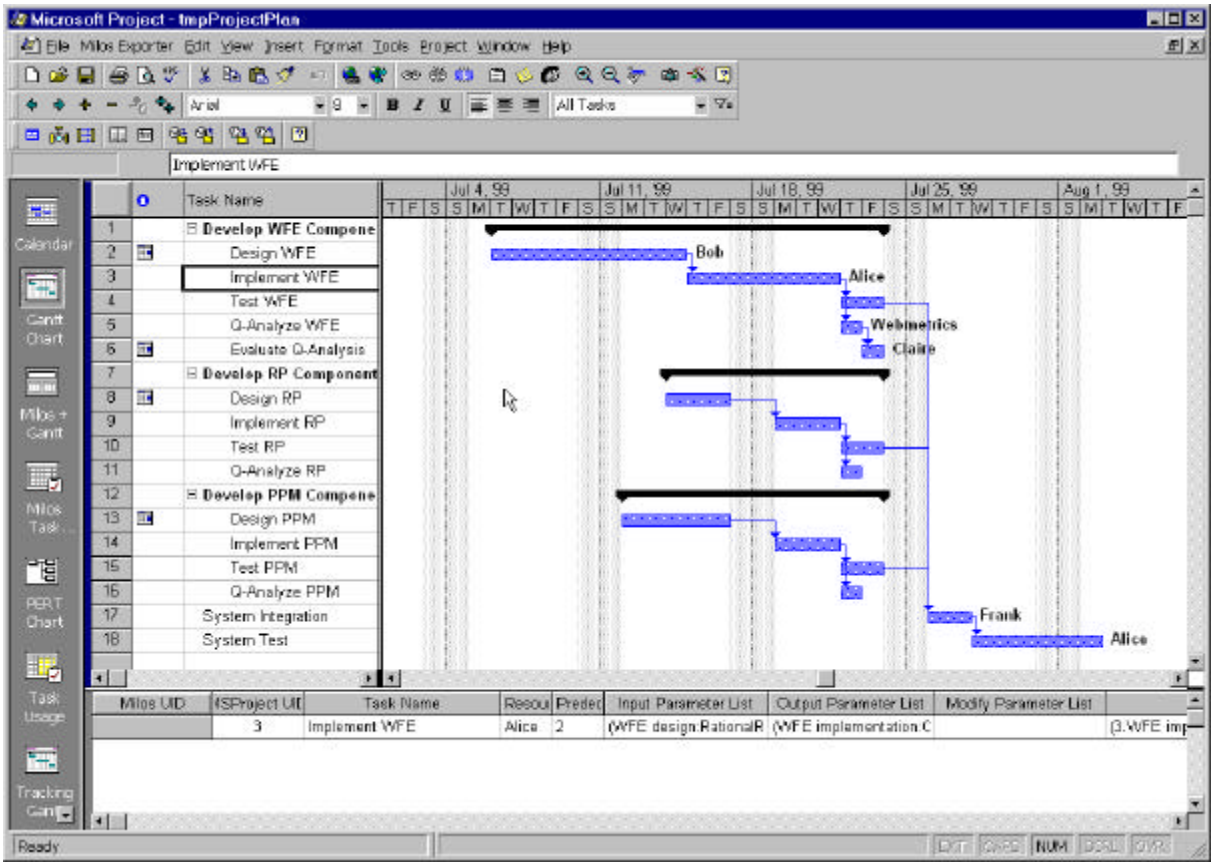

Figure 4: Example plan created with MS-Project.

Likewise, the planner proceeds with the remaining processes. The initial project plan is then uploaded to the COTS project planning tool MS Project. The screenshot taken from MS-Project in Fig. 4 shows a simplified example plan for developing the MILOS architecture. For each of the three components *Project Plan Management* (PPM), *Workflow Engine* (WFE) and *Resource Pool* (RP), the plan contains a task describing the component's development process. These tasks are complex, i.e. they are refined by a set of subtasks that describe the activities required to perform the task in more detail. As Fig. 4 shows, the complex task *Develop WFE component* consists of the subtasks *Design WFE*, *Implement WFE*, *Test WFE, Q-Analyze WFE*[1] and *Evaluate Q-Analysis*. Also shown is some scheduling information, i.e. planned start and finish times, duration as well as team members assigned to each task. In addition to the information shown in Fig. 4, the plan also contains a loop from *Test WFE* back to *Implement WFE*. This loop is modeled by specifying a product flow between those two processes using our extension to MS Project.

For plan enactment, the project planner exports the plan from MS-Project into MILOS. From now on, team members, regardless of their geographical location, can log into MILOS via standard Web browsers and are provided with individual workspaces. Figure 5 shows the current workspace of team member Alice. According to the project plan, she is responsible for the tasks *Implement WFE* and *System Test*. Consequently, these task appears on her to-do list.

[1] Quality analysis of code.

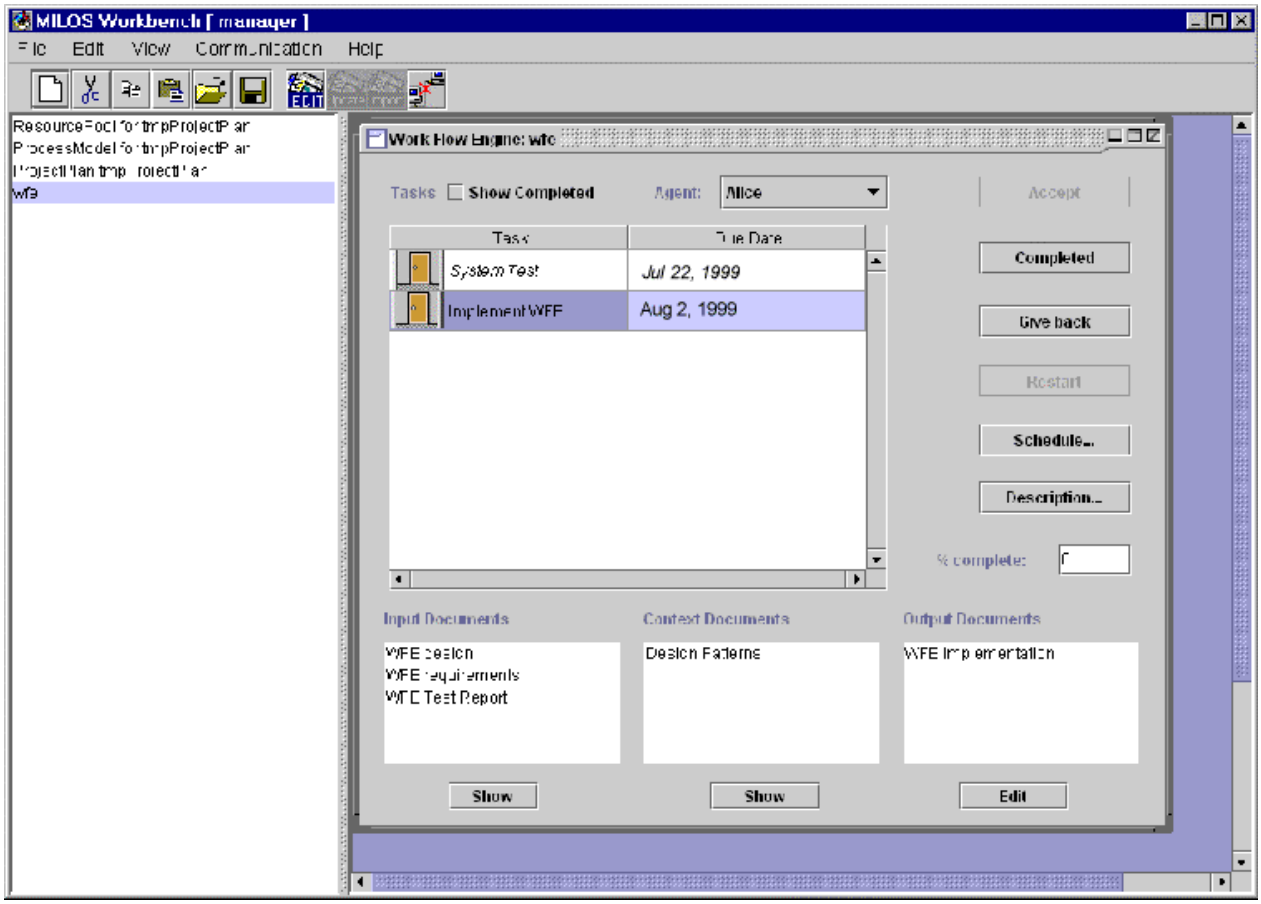

Figure 5: MILOS workspace for Alice.

The workspace allows Alice to browse the information associated with each task, e.g., a more detailed task description and scheduling information. In particular, she is given access to any documents needed to execute the task. Hence, the list of input documents (WFE design, WFE requirements, WFE Test Report) as well as the list of output documents (WFE implementation) is displayed for *Implement WFE*. In addition, the context document lists contains background information (here: a link to a Design pattern URL). The corresponding documents can be opened it with the appropriate tools (in this case: Rational Rose for the WFE design and MS Word for WFE requirements).

As soon as the required input document *WFE design* has been released by Bob, the team member who is responsible for task *Design WFE* (see Fig. 4), Alice is notified that the inputs of task *Implement WFE* are now available (Fig 6).

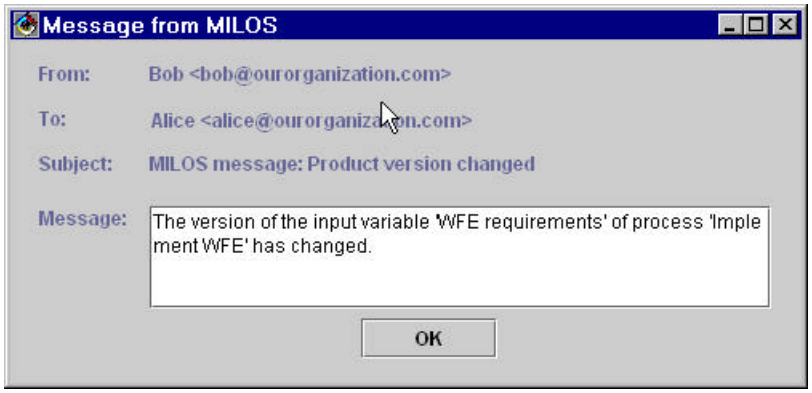

Figure 6: Input changed notification

After having inspected the design document, she forecasts her start and finish times for the implementation. In the case that her forecast violates the project schedule, the planner receives an automatically generated notification about this problem. However, since her forecast conforms to the schedule, no notification is sent.

After selecting the output document *WFE implementation*, a click on “Edit” starts the appropriate Java implementation environment for Alice. At the end of each day that she is working on this task, she can save her work and specify a "percentage complete" value for it. This value can be exported from MILOS back to MS-Project in order to provide the planner with up-to-date information on the project.

When she completes the task, it will be removed from her to-do list. In addition, the document *WFE implementation* is released, to the effect that the two succeeding tasks *Test WFE* and *Q-Analyze WFE* become executable. According to the plan (see Fig. 4), the former has not been assigned to any team member yet, while the latter has been assigned to a software agent (Webmetrics) that is a wrapper around a specific software metric tool. The software agent performs the task automatically as soon as it becomes executable.

Depending on the metrics gathered, the planner refines the task *Test WFE* to white-box testing. He defines two new subtasks *Write WFE Test Cases* and *Run WFE Test Cases* (Fig. 7).

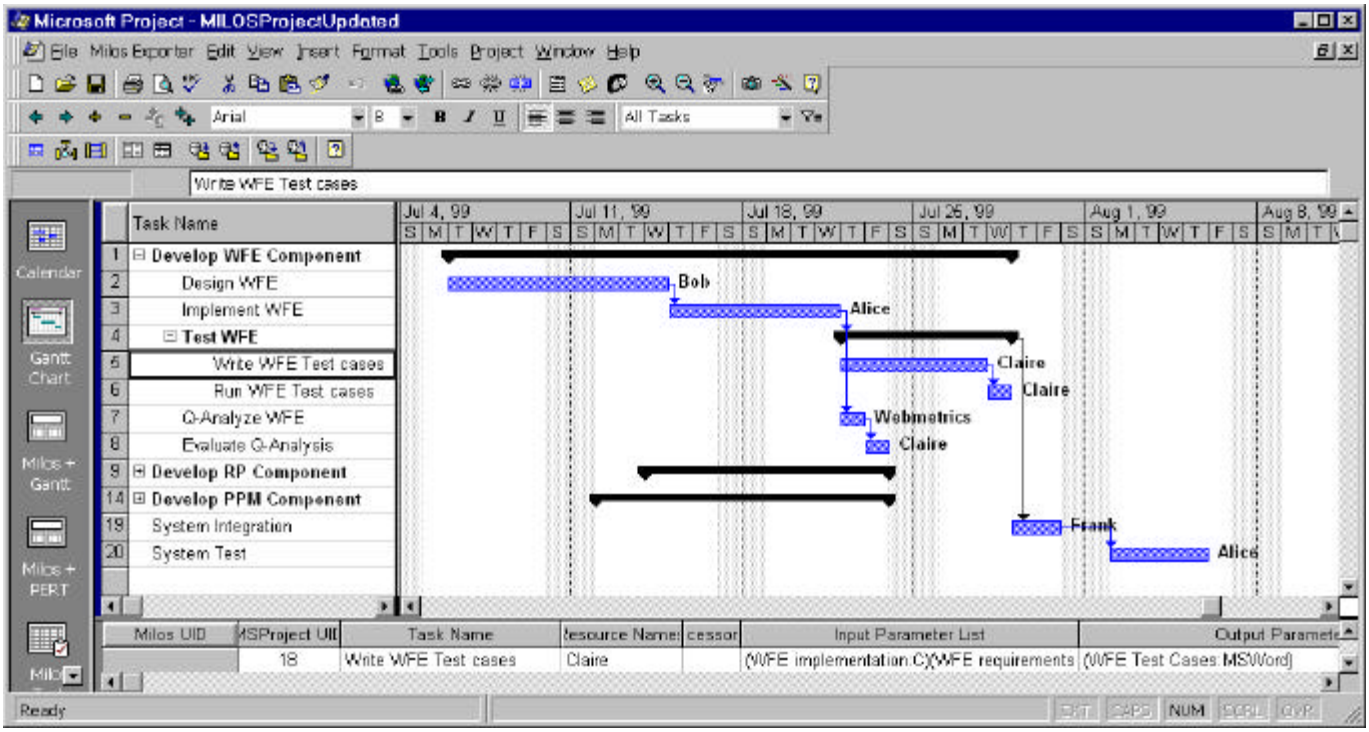

Figure 7: Updated project plan

After finishing the plan update, the planner exports it again into MILOS. This causes the two newly created tasks to appear on the to-do lists of team members the tasks were assigned to (here: Claire).

If Claire encounters problems with Alice's code during task *Run WFE Test Cases*, an MS-Word document containing a description of the problems will be created as output. The presence of this document will cause a notification to Alice stating that a new test report is available. Alice will then check this report and may then restart her implementation task. Alice will later release a new version of the document *WFE implementation* as soon as she has corrected the code. Analogous to the restart of task *Implement WFE*, the release of a new *WFE implementation* document version will cause a restart of the succeeding tasks *Test WFE* and *Q-Analyze WFE*. That way, a single restart might cause a "restart-cascade" that reaches all tasks affected by a change in a document.

While working on the implementation task, Alice is searching the Internet for information on Java documentation guidelines and finds the site “www.javaguide.com”. In addition, she queries the experience base for information on design documents with a similar complexity than the *WFE Design* to determine the approximate effort for implementing the system before she enters a forecast. She adds both information (URL and query) to the task.

In a next step, a member of the software process group analysis the project plan and finds out that Alice has entered additional information. She then decides to find more Java documentation guidelines and associates Alice’ URL and her own results to the *Implement* process in the library.

5. RELATED WORK

Related work comes mainly from two areas: Software Process Improvement and Knowledge Management. We first discuss software process improvement and we analyze some KM approaches.

Most process improvement approaches, e.g. capability maturity model, require describing the development processes more or less formally. Within the framework of software process modeling, several languages were developed that allow for describing software development activities formally (Osterweil, 1987; Curtis, Kellner, and Over, 1992, Armitage, and Kellner, 1994; Verlage, Dellen, Maurer, and Münch, J. 1996).

Software process models represent knowledge about software development. They describe activities to be carried out in software development as well as the products to be created and the resources & tools used. These models can be a basis for continuous organizational learning as well as the actual basis for the coordination and the management of the software engineering activities.

Understanding commonalties and differences between process types is a key factor for better process support. Process support includes improved communication, guiding people when performing processes, improving both processes and their results, and automating process steps (Rombach, and Verlage, 1995).

Software process modeling and enactment is one of the main areas in software engineering research. Several frameworks have been developed (e.g. procedural (Sutton, Osterweil, and Heimbigner, 1995], rule-based (Kaiser, Feiler, and Popovich, 1988; Peuschel, Schäfer, and Wolf, 1992], Petri net based (Bandinelli, Fuggetta, and Grigolli, 1993a], object-oriented (Conradi, Hagaseth, Larsen, Nguyen, Munch, Westby, Zhu, 1994)).

Managing software process knowledge is also the goal of the experience factory approach (Basili, 1989; Basili, Gianluigi, Caldiera, and Rombach, 1994). They distinguish between the organizational structure to manage the software knowledge (the experience factory department) and the activities that have to be carried out to build an experience factory.

Knowledge management and organizational memory is currently a hot topic in research. People from several areas (e.g. for economics see (Davenport, Jarvenpaa, and Beers, 1997), for knowledge engineering see (Wielinga, Sandberg, and Schreiber, 1997)) are working on it.

The use of case-based reasoning technology for experience management in software engineering is discussed in (Althoff, Bomarius, and Tautz, 1998). Their approach is not directly linked to project execution support and therefore fails in providing an operational feedback loop between project execution and organizational learning. Their approach could be easily integrated with MILOS by pointing our CBR queries to their system.

The expert system group at the German DFKI is also following a process-centered approach to knowledge management. Their tool is based on a business process modeling approach and is, compared to MILOS, fairly inflexible at enactment time: changing the process by simply replanning the project with a COTS tool is not supported. On the other hand, they are supporting retrieval based on doain ontologies (Kühn and Abecker, 1997) which provides more semantic than normal information retreval approaches.

Ontology-based retrieval is also investigated by (Decker, S., Erdmann, M., Fensel, D., and Studer, R., 1999). Ontology-based KM approaches are not providing a process-centered approach and therefore require the user to specify queries for a given task (whereas MILOS can provide users with predefined queries for tasks). A similar argument holds for hypertext-based approaches (e.g. (Euzenat 1996)): they do not provide task-oriented access to knowledge but force the user to navigate to it.

6. CURRENT AND FUTURE WORK

The MILOS system is implemented in Java using the OODBMS GemStone as the application server. The MS Project integration is developed with MS Visual Basic. A single-user version was demonstrated on the International Conference on Software Engineering in Los Angeles in May 1999.

We are currently working on closing the feedback loop by updating process models based on project plans and on extending the facilities for linking background knowledge to processes (CBR/SQL is not yet integrated).

We are planning to evaluate the approach by applying it to our own software development process in the future.

Beside using CBR and standard SQL for accessing background information, we are thinking about using an ontology-based approach for more formalized knowledge representation.

7. CONCLUSION

In this paper we described the MILOS approach for supporting learning software organizations. The MILOS system is an Internet-based process-centered knowledge management environment. The environment structures knowledge around development processes: In the center of our representation are processes. We believe that the process-centered structure is also applicable in other application areas. For example, we used a similar approach for urban land-use planning in the past.

Linked to a process, the user can find methods (describing ways how to perform the process to reach its goals), products (input and outputs to the process), factual knowledge in the form web references and predefined queries, and knowledge about the qualifications needed to perform the process.

The process-centered structure of the system has the following advantages:

- Processes are “natural“ entities for managers and team members: they are well used to thinking in processes (e.g. for project planning).

- For their daily work, people don’t need knowledge per-se but knowledge for performing specific tasks. A process-centered knowledge management system associates explicitly the task with the knowledge needed for it.

By linking web references to task, the lost in hyperspace problem is reduced because the user immediately finds the knowledge needed instead of being forced to browse to relevant pages.

A process engine that guides the human users in their daily work interprets the explicit description of processes. This guidance is especially useful for new employees because they lack the knowledge about the standard procedures of a company.

The feedback cycle creates a learning software organization: our approach allows to package “good” elements of successful project plans into reusable process models and, hence, supports the implementation of a continuous process improvement strategy for software companies.

8. ACKNOWLEDGEMENTS

The work on the MILOS system was supported by NSERC, The University Of Calgary, Nortel and DFG with several research grants.

We would like to thank Barbara Dellen, Fawsy Bendeck, Sigrid Goldmann, and Boris Koetting for their valuable input. The user interfaces of MILOS were implemented by Mike Gao.

9. REFERENCES

Althoff, K.-D. & Bomarius, F. & Tautz, C. (1998). Using Case-Based Reasoning Technology to Build Learning Software Organizations. In Proceedings of the 1st Interdisciplinary Workshop on Building, Maintaining, and Using Organizational Memories (OM-98), 13th European Conference on AI (ECAI’98), Brighton, http:// SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-14/

Armitage, J., and Kellner, M. (1994). A conceptual schema for process definitions and models. In D. E. Perry, editor, Proceedings of the Third International Conference on the Software Process, IEEE Computer Society Press.

Bandinelli, S., Fuggetta, A., and Grigolli, S. (1993). Process Modeling-in-the-large with SLANG. In IEEE Proceedings of the 2nd International Conference on the Software Process, Berlin (Germany).

Basili, V. R. (1989). The Experience Factory: packaging software experience. In Proceedings of the Fourteenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt MD 20771.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). Experience Factory. In Encyclopedia of Software Engineering (J. J. Marciniak, ed.), vol. 1, John Wiley Sons.

Conradi, R., Hagaseth, M., Larsen, J. O., Nguyen, M., Munch, G., Westby, P., and Zhu, W. (1994). EPOS: Object-Oriented and Cooperative Process Modeling. In PROMOTER book: Anthony Finkelstein, Jeff Kramer and Bashar A. Nuseibeh (Eds.): Software Process Modeling and Technology, 1994. Advanced Software Development Series, Research Studies Press Ltd. (John Wiley).

Curtis, B., Kellner, M., and Over, J. (1992). Process modeling. Communications of the ACM, 35(9): 75–90.

Davenport, T.H. & Jarvenpaa, S.L. & Beers, M.C. (1997). Sloan Management Review, 37 (4):53-65, Improving Knowledge Work Processes,1997.

Decker, S., Erdmann, M., Fensel, D., and Studer, R.(1999). Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al. (eds.), Semantic Issues in Multimedia Systems, Kluwer Academic Publisher, Boston.

Dellen, B., Kohler, K., and Maurer, F. (1996). Integrating Software Process Models and Design Rationales. In Proceedings of Knowledge-Based Software Engineering Conference (KBSE-96), IEEE press.

Euzenat, J. (1996). Corporate Memory through Cooperative Creation of Knowledge Bases and Hyper-documents. In Proceedings of the 10th Knowledge Acquisition, Modeling and Management for Knowledge-based Systems Workshop (KAW'96), Banff.

Kaiser, G. E., Feiler, P. H., and Popovich, S. S. (1988). Intelligent Assistance for Software Development and Maintenance, IEEE Software.

Kühn, O. & Abecker, A. (1997). Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges. In Journal of Universal Computer Science 3, 8, Special Issue on Information Technology for Knowledge Management, Springer Science Online. URL: http://www.iicm.edu/jucs_3_8/ corporate_memories_for_knowledge.

Maurer, F., Dellen, B. (1998). A concept for an Internet-based process-oriented knowledge management environment, in: Gaines, B.R., Musen, M.: Proc. 11th Banff Knowledge Acquisition Workshop, Banff, Alberta, Canada.

Osterweil, L. (1987). Software Processes are Software Too. In Proceedings of the Ninth International Conference of Software Engineering, Monterey CA, pp. 2-13.

Peuschel, P., Schäfer, W., and Wolf, S. (1992). A Knowledge-based Software Development Environment Supporting Cooperative Work. In: International Journal on Software Engineering and Knowledge Engineering, 2(1).

Rombach, H.-D., and Verlage, M. (1995). Directions in software process research. In M. V. Zelkowitz (Eds.), Advances in Computers, vol.41. Academic Press.

Verlage, M., Dellen, B., Maurer, F., and Münch, J. (1996). A synthesis of two software process support approaches. In Proceedings 8th Software & Engineering and Knowledge Engineering (SEKE-96), USA.

Wielinga ,B.J. & Sandberg, J. & Schreiber, G. (1997). Methods and Techniques for Knowledge Management: What has Knowledge Engineering to Offer, Expert Systems with Applications 13, 1, 73-84.