

Process-centered Knowledge Organization for Software Engineering

Frank Maurer

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

Harald Holz

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

From: AAAI Technical Report WS-99-10. Compilation copyright © 1999, AAAI (www.aaai.org). All rights reserved.

Abstract

This paper argues for using process models as the primary means to structure and organize knowledge in software organizations. We discuss how process models can support project planning and illustrate how a process-centered organization of knowledge helps the individual developer to find appropriate background information for his current task during process enactment. The paper concludes with an overview on our MILOS system: a process-centered, Internet-based, knowledge management and process support environment.

Introduction

Software development is a knowledge-intensive process where highly educated people have to cooperate to reach a business goal. Managing software process and software artifact knowledge properly is beneficial to a company because it supports software process improvement initiatives. 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 commonalities 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.

Process-centered Knowledge Organization

Our approach distinguishes between three kinds of knowledge:

- Generic, reusable knowledge,
- Knowledge about a specific project plan, and
- 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. That implies that processes are in the core of the knowledge organization and are our primary means for indexing information. We use our process modeling language MILOS as our primary way to structure software development knowledge.

Generic Process Models are reusable process models. They associate generic knowledge to entities of the process model. Knowledge may be stored in several forms:

- Concrete knowledge chunks for human use are stored as context information: URL references pointing to HTML pages or other files containing information in specific document formats, e.g. Checklists for reviews or manuals for specific tools.
- 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:

Database queries: To access knowledge stored in a relational database, the process model associates

predefined SQL queries with tasks¹. When a user carries out the task, he will be able to click on the query. Then the system will connect to the database, query it and present the results to the user. An example for such knowledge is a query that determines the average time spent on 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.

Information retrieval requests: 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. 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.

Project Plans adapt generic process models to the needs of a concrete project. This tailoring 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 the method of choice.
- standard project planing: Assigning tasks to responsible agents, scheduling tasks, cost estimation etc.

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

- how to determine that a generic model can be reused in a given situation
- how to support the customization process.

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 is handled by a flexible workflow/process engine. This manages the state of the work process and its tasks, to-do lists for its users, the products created during process enactment, traceability relationships between process entities (Dellen, Kohler, Maurer 1996), etc. 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).

Integrating Process Modeling, Project Planning & Enactment

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

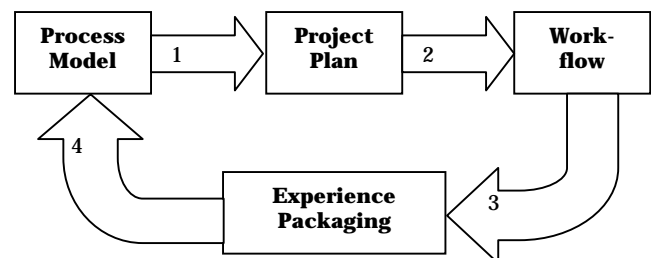


Figure 1: Knowledge Utilization Cycle

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) or changed using our project planning editors. 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

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

project plan is the basis for our workflow engine (MILOS 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 finishing a task, the outputs are accessible as input in the successor tasks. We are currently working on the query functionality and the feedback cycle. A user will then be able to select parts of the project plan (3) and generate an equivalent partial model in the experience factory that stores all process models (4).

The MILOS System

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

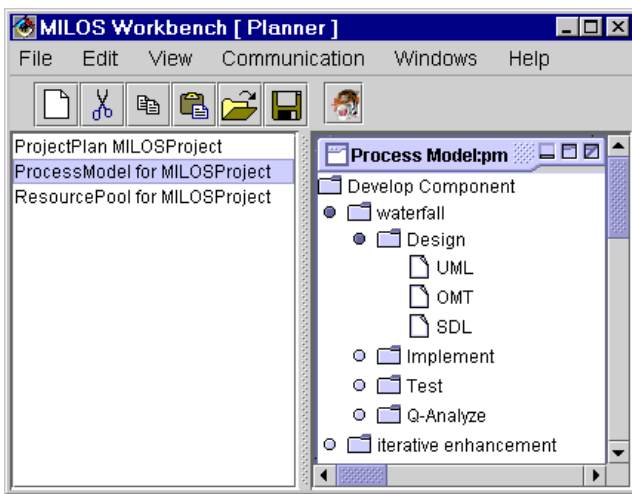


Figure 2: Example process model.

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”).

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.

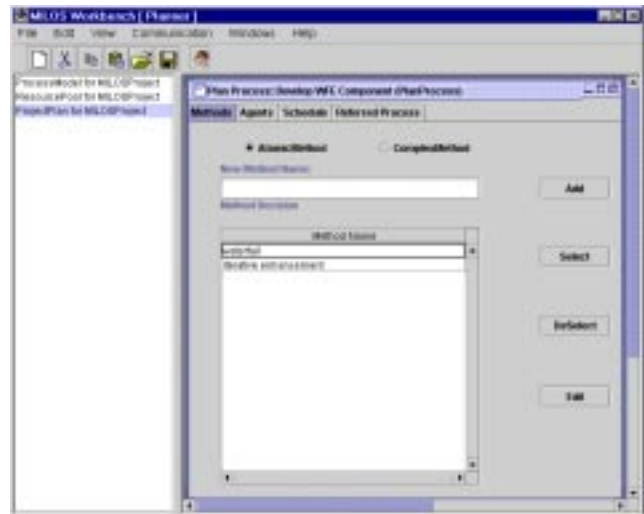


Figure 3: Planning decision.

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 is familiar 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

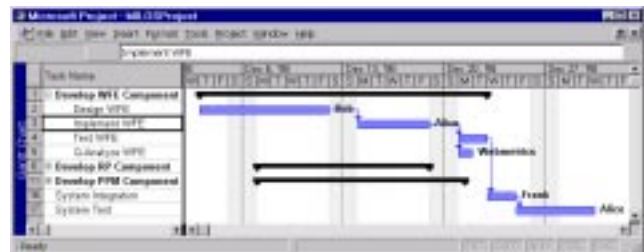


Figure 4: View on partial plan with MS-Project.

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.

Likewise, the planner proceeds with the remaining processes. The screenshot taken from MS-Project in Fig. 4 shows a partial plan for developing the MILOS system. 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 MS-Project's additional task attribute fields.

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 task *Implement WFE* and, consequently, this task appears on her to-do list.

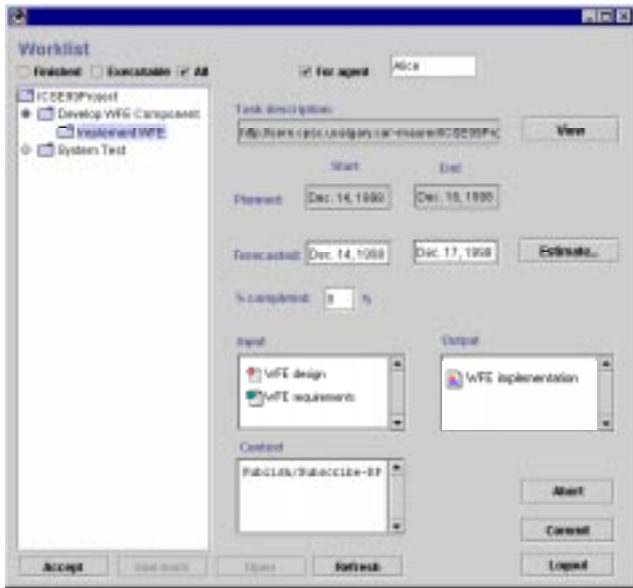


Fig. 5: MILOS workspace for Alice.

The workspace allows Alice to browse the information associated with each task, e.g. a more detailed task description (including the task hierarchy) and scheduling information. In particular, she is given access to any documents needed to execute the task. Hence, the list of input documents (the design document and the requirements document) as well as the list of output documents is displayed for task *Implement WFE*. These lists can be specified in the project plan. A double-click on a document opens it with the appropriate tool.

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 task *Implement WFE* has become executable. In addition to the input documents specified in the plan, Bob added the document *Publish/Subscribe-Pattern* as a context URL to Alice's task. Thus, during the implementation Alice will be given easy access to important information that helps her to better understand both the design and related implementation issues.

Before she starts working on the task, Alice lets the system estimate its duration. This estimate is based on the number of classes/methods in the design document and Alice's average coding speed (retrieved by a query associated with the task). According to this estimate, she forecasts her start and finish times for the implementation to be the current

date (Dec. 14) and Dec. 17, respectively. In the case that her forecast violated the project schedule, the planner would receive an automatically generated email notification about this problem. However, since her forecast conforms to the schedule, no notification is sent.

A double-click on the (empty) output document *WFE implementation* 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 will be exported from MILOS back to MS-Project in order to provide the planner with up-to-date information on the project.

When she is finished with 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 specific metric tool. This tool acts as a software agent that performs the task automatically as soon as it becomes executable.

Depending on these measurements, the planner might want to refine the task *Test WFE* to either a black-box or white-box testing process. In our example, the planner settles for white-box testing because of a high number of conditional expressions in Alice's code. Hence, he refines the task *Test WFE* by creating two new subtasks *Write WFE Test Cases* and *Run WFE Test Cases*. In addition, he schedules these two new tasks and updates his former estimate on the finish time of the task *Test WFE*. Because this former estimate was based on his optimistic assumption that black-box testing would suffice, the planner now allocates more time for task *Test WFE*. As a consequence, the schedule for the succeeding tasks *System Integration* and *System Test* also has to be changed.

When he is finished with updating the project plan, the planner exports it again into MILOS. This causes the two newly created tasks to appear on the to-do lists of those team members the tasks were assigned to. In addition, the team members responsible for the tasks whose time scheduling had to be changed receive a corresponding notification. This allows them to update their own work schedule, in particular their forecasts on start and finish times.

In case any problems with Alice's code should be encountered 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 restart of the task *Implement WFE*, i.e. it will appear once again on Alice's to-do list, together with the problem report as an additional input to the task. However, since by now the calendar has advanced to Dec. 28, whereas the implementation task was scheduled to finish by Dec. 18, the planner will receive an automatically generated email about this delay. That way, he will have the opportunity to correct the plan in time if project deadlines make this update necessary.

Meanwhile, Alice will 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.

Discussion

In this paper we described the MILOS system, an Internet-based process-centered knowledge management environment. The environment structures knowledge around work processes: In the center of our representation are processes. 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 of product instances (often implemented as web references), 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: we provide a tool that allows to package "good" elements of successful project plans into reusable process models. This technology supports the implementation of a continuous process improvement strategy for software companies.

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.

References

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