Process Support for Distributed Extreme Programming Teams

Frank Maurer
University of Calgary
Department of Computer Science
Calgary, Alberta, Canada, T2N 1N4
+1 (403) 220 3531
maurer@cpsc.ucalgary.ca

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

ABSTRACT

Extreme programming (XP) is arguably improving the productivity of small, co-located software development teams. In this paper, we described an approach that overcomes the XP constraint of co-location by introducing a process-support environment (called MILOS) that helps software development teams to maintain XP practices in a distributed setting. MILOS supports project coordination, information routing, team communication, and pair programming.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *programming teams.*

General Terms

Management

Keywords

virtual software development teams, distributed extreme programming, process support

1. INTRODUCTION

Extreme programming (XP) [3, 4, 9] is one of the most innovative software development approaches of the last years. The XP movement seems to be driven by disappointment with current software development practice: low productivity and low user satisfaction are seen as commonplace. Software development teams are often delivering huge amounts of documentation (for example requirements specifications, system architecture descriptions, software design documents, test plans) instead of delivering useful functionality to the client. Sometimes, projects are cancelled before the system is deployed – wasting all the effort that was already spent on analysis and design.

XP, on the other hand, focuses development effort on activities that deliver high-quality functionality to the end user as fast as possible. Deliverables usually are restricted to high-level use cases (user stories), source code and test code. XP has in its original form proposed by Beck [3] two severe limitations. First, it does not scale well to larger teams. Second, it requires the XP team to be collocated. Overcoming the collocation requirement while preserving the high productivity and quality of XP processes is one goal of our approach and the focus of this paper.

In Section 2 we give an overview on our MILOS approach. Section 3 and 4 provides a detailed usage scenario while section 5 describes the state of implementation. We conclude with a summary and a look on future work.

2. THE MILOS APPROACH

The overall goal of the MILOS approach is to support process execution and organizational learning for virtual software development teams. In this paper, we focus on how MILOS supports Distributed XP (DXP). [8] describes the knowledge management aspect in more detail.

The support provided by MILOS should be minimally intrusive to reduce overhead: MILOS stands for "**M**inimally **I**nvasive **L**ong-term **O**rganizational **S**upport". The MILOS approach can be applied for open source projects as well as for commercial teams that are distributed over the world. It was adapted to support Distributed XP.

We now describe requirements that were underlying the development of MILOS. Then we explain the overall structure of the approach.

2.1 Requirements on Tool Support for Virtual Teams

Using XP and open source processes as a baseline, the work process of virtual software teams can be improved in several ways.

Project coordination: XP teams are usually much more closely coordinated than open source projects. Hence, project coordination support is strongly required for DXP. This should allow the team to assign tasks to developers, set deadlines and get an overview on the current state of the project. Team members on the other hand should be able to access their to-do lists and retrieve relevant information for performing their tasks easily.

Synchronous communication: XP replaces documentation by synchronous, face-to-face communication. Face-to-face communication is not feasible in a distributed team and needs to be replaced by technical means. Besides using e-mail for communication, synchronous communication like audio and video calls or text chat may be helpful. If two developers want to do pair programming, application sharing is needed.

Active notifications and information routing: Instead of merely making information available for pull access, it would be useful push to important information to the users as soon as it becomes accessible. This may help to overcome the missing informal communication that happens in coffee breaks & lunches or by simply overhearing conversations in the pair programming area of an XP project. This push approach should include notifications when important events occur in a project. For example, a manager needs to be notified when a task gets delayed or a developer needs to be notified when an update of another component becomes available that she is using. The change notification mechanism of MILOS is discussed in [7].

Integrate process execution with knowledge management: In virtual teams, members frequently change. Hence, there is a high demand on bringing new members up to speed on their tasks and in preserving good sources of knowledge for the organization. As software development often has to struggle with fast changing technology, keeping the contents of an experience base up to date is a demanding task and needs to be integrated as much as possible with the everyday processes of executing processes. While XP relies primarily on face-to-face communication for knowledge exchange, the MILOS approach to DXP includes means for building a process centered experience base for the team.

MILOS provides overviews on the current state of all tasks of a project as well as project management queries to find late tasks. MILOS also allows accessing the information produced as the output of a task easily.

Team members are able to access the project plans and the information related to each task using a standard Web browser.

3. USING MILOS DXP FOR DISTRIBUTED EXTREME PROGRAMMING

The following scenario illustrates the infrastructure provided by the MILOS framework to support distributed extreme programming. Team members access the Internet with a Web browser and connect to the MILOS server. First, they login to the MILOS system to access their workspace. From their workspace they may retrieve the list of current projects, user stories, currently available tasks, task estimation, and pair programming facilities. Pair programming is supported via NetMeeting. We now illustrate MILOS using an example taken from [3].

3.1 Creating User Stories

After the creation of an initial project and the assignment of a project manager to it, the customer is ready to enter story cards into the MILOS system (see Figure 1).

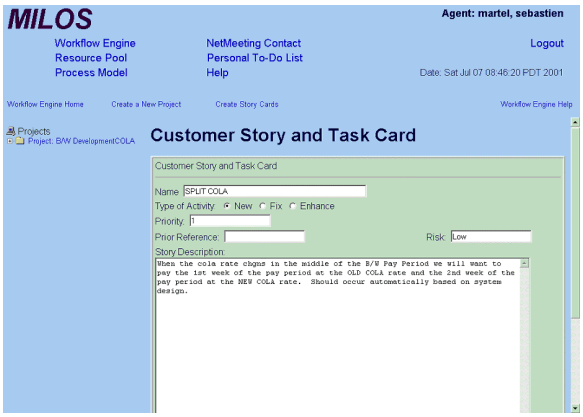

Figure 1. Story Card

The top part of the main screen displays a menu that allows accessing all components of MILOS (Workflow Engine, Resource Pool, Process Model, etc). Below that a menu is displayed that allows for manipulating the selected component (here, is allows to access the workflow execution support component). The left side shows the hierarchical task decomposition of all projects. Selecting a task in the task decomposition will display detailed task information on the right side. The programmers can then contact the customer, either through the MILOS framework or by conventional means, and discuss the story with them if need be. They can revise the description of the user story – creating a new version of the existing card. In addition, they can then add notes to the story card pertaining to implementation details and split up the story into several smaller stories if the scope is too large. They would then proceed to decompose the story into specific programming task that will be needed to satisfy the next build. The workflow engine of MILOS handles the creation and changes of tasks.

3.2 Task Creation

For each user story, the MILOS system automatically creates a top-level process "Design and implement user story <story number>". The input of this process is the newly created user story. Then the programmer can decompose the user story into smaller and more concrete tasks. A possible decomposition of the above user story could have two separate sub-tasks (see Figure 2).

Given the user story, the programmer may create a sub-task called "create m/frame" and another one called "create RM boundary".

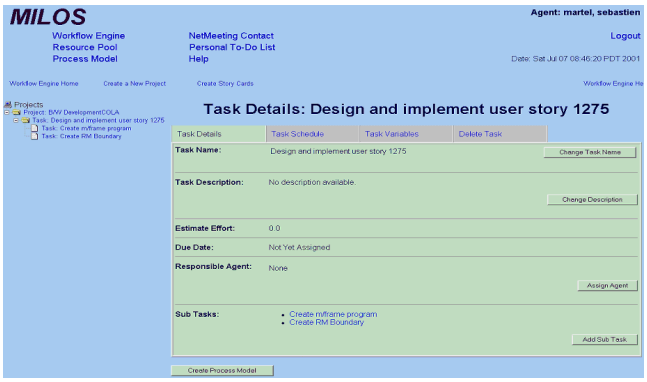

Figure 2. Task Decomposition

After having decomposed user stories into concrete programming task, the programmer may describe the task in more detail using the workflow engine user interface. Tasks are associated with specific projects and can be assigned to various team members. For each task, the manager enters planned start and end dates. In addition, the users are able to define the inputs and outputs of processes. The story card automatically becomes an input of all subtasks. Furthermore, the users are able to specify the information flow between tasks by defining the output of one process to become the input of another (see below).

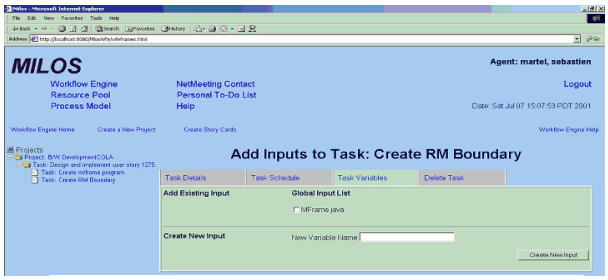

Figure 3. Defining the inputs of a task

Specifying the information flow allows the MILOS system to provide access to input information that was created as the output of another task: the output of a task, e.g. a source code file, is transferred to the MILOS server and stored in a version management system. From there, any successor task may access the current version as well as older versions. As this is done via HTTP requests, tunneling through a firewall usually is not a problem. The programmer is able to estimate the effort and the forecasted end dates (see Figure 4). The effort simply consists of the total number of workdays needed to complete the task (measured in ideal engineering time). To set the forecasted end date, the developer takes the required effort as well as his overall workload into account.

The team lead can keep a close eye on the progress of each task by watching the "percentage complete" values and steer the team in the correct direction if the requirement for the next build will not be met. After having signed-up for some task, a programmer can pair with another programmer through the MILOS framework.

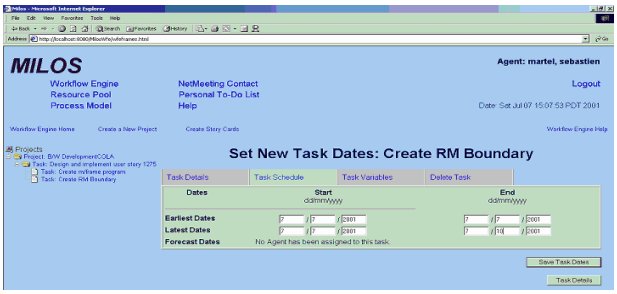

Figure 4. Estimating effort needed

4. PAIR PROGRAMMING

Using Microsoft NetMeeting, MILOS provides an audio and video link between two developers and the ability to share the desktop between them. These capabilities are used to support pair programming in a distributed setting. The MILOS system keeps track of who is logged on to the system and provides the possibility to contact the responsible team member for a task or any other team member that is currently logged in. Figure 5 is a screenshot of shared desktop using NetMeeting. The screen shows the MILOS environment in the back, NetMeeting on the top right and a VisualAge for Java window at the top left. The programmer (sitting at a remote machine) just enters a method definition. The local team member inspects the code and can comment on it using audio/video conferencing. The local programmer may also take over and edit the method from his machine.

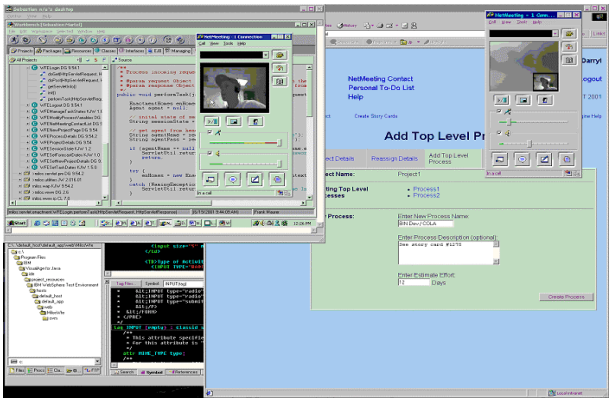

Figure 5. Pair programming with NetMeeting

The screen might seem cluttered at first. However, we tried to show as much features as possible. Normally a programmer would only look at the other desktop, see the top left corner, and switch between screens when needing to look up a function definition or when video conferencing. After the pair programming team has completed a task, they can update the task status and mark it as completed. They can also upload any output

files to the MILOS server that, in turn, would route them to other team members who would need them.

4.1 Initial Results and Lessons Learned

We were using MILOS for our own development processes over the last time. Although we do not yet have statistical valid data, we can provide some initial results. Even though we did not quantitatively measure the gain in productivity offered by distributed extreme programming at this time, we can provide the following insight as to its advantage.

- Overall, we were able to apply XP in a distributed setting.
- Having the computing power of two machines while having the impression of only using one is sometime useful. Since NetMeeting is only bandwidth intensive and does not use much CPU time the person receiving the shared desktop can use their CPU to regenerate code or compiling a new build.

However, we also ran into some problems attributed to various hardware and software technologies.

- Due to network latency, it is better for the programmer that is typing to share their desktop. This even holds for high-bandwidth connections. When switching position, the other programmer can share their desktop. This allows smoother programming since there is no delay for the programmer who is actually typing.
- When pair programming with different screen size and resolution the pair should find a resolution that is comfortable for both developers. If the two resolutions differ by a too great amount, one of the programmers will need to use the scrollbars extensively.
- Using video conferencing AND audio on a dial up modem (56K) is not practical. The team should use broadband access or only audio conferencing with team members that only have access to dial-up Internet.
- The video link is often unnecessary if the pair programming together is already acquainted.
- Color, fonts and sound scheme needs adjustment in order to be properly viewed when sharing desktops. For example, unless having access to a high bandwidth connection enabling "sharing in true color mode" is not practical.
- There is no need to acquire video equipment able to capture 40 frames per second in 600x800 resolution when the link to your teammates can only sustain a bandwidth of 40KB/sec.
- NetMeeting is restricted to 1 to 1 audio/video communication. Using tools like the Microsoft Conference Server would provide the necessary support for multiple client videoconferencing.

5. STATE OF IMPLEMENTATION

MILOS is Web-based and accessible as a web service on the MILOS web site[1] from any machine connected to the Internet using a standard Web browser. MILOS DXP uses the EJB-based version as its basis and adds its extensions.

All the core functionality described in the paper is implemented so far that we were using MILOS for our own development processes. Nevertheless, the system still has some bugs and we are currently (April 2002) stabilizing the implementation as well as improving its usability by using feedback from our development team.

MILOS and MILOS DXP are open source software that can be downloaded from the MILOS Web site.

6. RELATED WORK

Related work comes mainly from two areas: Software Process Support and (Distributed) Extreme Programming. As we already discussed XP and DXP, we focus here on related work in process support.

Most process improvement approaches, e.g. capability maturity model, SPICE, QIP, 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 [1, 6, 11]

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.

Software process modeling and enactment is one of the main areas in software engineering research. Several frameworks have been developed (e.g. procedural [11], rule-based [10, 12], Petri net based [2], object-oriented [5]).

Process modeling and enactment approaches usually are used to rigorously define heavy-weight processes. They are weak concerning light-weight approaches like XP and do not directly support key XP practices. They also are not good at providing a good communication and collaboration infrastructure for virtual teams.

7. CONCLUSION AND FUTURE WORK

In this paper, we described our approach for supporting virtual software teams in distributed extreme programming. The MILOS system is an Internet-based process-centered process support environment that supports communication, collaboration and coordination of DXP teams.

With MILOS DXP, we are aiming at an improved efficiency of virtual teams. Whereas undoubtedly the introduction of new tools

1 http://sern.ucalgary.ca/~milos

at first results in an increased workload, we argue that, in the long run, the proposed approach will improve productivity of virtual software development teams (although it will most probably not reach the same productivity levels as collocated teams).

Our future work will focus on four aspects:

- Stabilizing the MILOS implementation and usability improvements
- Formal evaluation of the approach
- Extreme federations
- Knowledge management for DXP

Stabilizing the MILOS implementation and usability improvements: After porting MILOS to WebSphere Server 4.0, we encountered some bugs and instabilities. As we were using MILOS for our own process support, we were experiencing some sub-optimal user interaction. We are planning to fix these this summer (Summer 2002). MILOS is offered as a Web-based service to the software development community. We expect to get valuable feedback from MILOS users to determine future improvements.

Formal evaluation of the approach: We would like to set up controlled experiments to evaluate the feasibility and the benefits & problems of distributed extreme programming. In addition, we would like to compare the productivity and quality of XP teams and DXP teams to determine the influence of collocation on productivity.

Extreme federations: One of the problems of XP is scalability concerning team size: XP works for small teams of five to ten people but there is some doubt that it works with even a mid-sized team of twenty people. One way to scale it up could be to have loosely coupled federations of XP teams that work together on a single project. This poses several interesting research questions:

- How can we preserve XP productivity and quality in multi-team environment?
- Do we need additional documentation and, if so, how much more? And what needs to be documented to enable a smooth work of the Extreme Federation.
- Do Extreme Federations needs a component architecture to work?
- How fixed need the interfaces between components of individual XP teams be? How much flexibility and/or adaptability of requirements do Extreme Federations loose compared with “normal” XP teams?

Knowledge management for DXP: XP is very weak in conserving the knowledge gathered by the development team. It’s focus on verbal communication for knowledge exchange makes it difficult to preserve information in a storable format. As a result of keeping development knowledge primarily in the heads to the people, XP will run into trouble when the members of the development team change frequently or when the development on the system stops for some time and is then resumed. Hence, an approach is needed that integrates knowledge management and DXP.

8. ACKNOWLEDGEMENTS

The work on MILOS DXP was supported by the NSERC (National Science and Engineering Research Council of Canada), ASERC (Alberta Software Engineering Research Consortium), The University Of Calgary, and Nortel Networks with several research grants.

We specifically would like to thank Sandra Barlot, Subhendu Chattopadhyay, Darryl Gates, Christine Jia Li, Raul Nemes, Philip Nour, and Jay Wallace for implementing and testing the MILOS DXP system.

9. REFERENCES

[1] Armitage, J., and Kellner, M. (1994). *A conceptual schema for process definitions and models*. In D. E. Perry, editor, Proc. of the Third Int. Conf. on the Software Process, IEEE Computer Society Press.

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

[3] Beck, K.: *Extreme Programming Explained: Embrace Change*, Addison-Wesley Pub Co, 1999, ISBN: 0201616416

[4] Kent Beck, Martin Fowler: *Planning Extreme Programming*, Addison-Wesley Pub Co, 2000, ISBN: 0201710919

[5] Conradi, R., Hagaseth, M., Larsen, J. O., Nguyen, M., Munch, B., Westby, P., and Zhu, W. (1994). *Object-Oriented and Cooperative Process Modeling in EPOS*. 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).

[6] Curtis, B., Kellner, M., and Over, J. (1992). *Process modeling*. Comm. of the ACM, 35(9): 75–90.

[7] Dellen, B.: Change Impact Analysis Support for Software Development Processes, Ph.D. thesis, University of Kaisersalutern, Germany, 2000

[8] Holz, H., Könnecker, A., Maurer, F.: *Task-Specific Knowledge Management in a Process-Centred SEE*, Proceedings of the Workshop on Learning Software Organizations LSO-2001, Springer, 2001.

[9] Jeffries, R., Anderson, A., Hendrickson, C.: *Extreme Programming Installed*, Addison-Wesley Pub Co, 2000, ISBN: 0201708426

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

[11] Osterweil, L. (1987). *Software Processes are Software Too*. In: Proc. of the Ninth Int. Conf. of Software Engineering, Monterey CA, pp. 2-13.

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