# Supporting Distributed Extreme Programming

Frank Maurer

University of Calgary
Department of Computer Science,
Calgary, Alberta, Canada, T2N 1N4
maurer@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 tries to overcome the XP constraint of co-location by introducing a process-support environment (called MILOS for Agile Software Engineering - MILOS ASE) that helps software development teams to maintain XP practices in a distributed setting. MILOS ASE supports project coordination using the planning game, user stories, information routing, team communication, and pair programming.

## 1 Introduction

Extreme programming (XP) [3][4][10] is one of the most innovative software development approaches of the last years. The agile 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 user. 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 user as fast as possible. Documentation is often restricted to high-level use cases (user stories), source code and test code. XP is based on several basic assumptions and trade-offs. The two that directly influence our work are:

- *Nobody can always predict the future correctly*: Many standard software engineering practices are based on the 1970'ies observation that resolving a problem in a software system gets much more expensive the later it is found. Software engineering textbooks (e.g. [18]) claim that fixing a problem after release of the software is 60-100 times more expensive than in the definition phase. As a result, software engineering processes often try to minimize the number of problems found in later development phases by spending much effort in earlier stages: by writing very detailed requirement and design documents the software development team is trying to make sure that the future system fits the user's needs (requirements are met) and that the system functions properly (adequate design). As a result, it usually takes quite some time (several months or years) to deliver productivity-increasing software function-

ality to the end users. The drawbacks of this approach are twofold. First, it increases the risk that the system is obsolete when it is introduced because the business environment has changed. Second, it assumes that somebody is able to predict what functionality will be required in the future (at the time when the system is deployed). XP and other agile methods, on the other hand, try to put a useful system in place within a very short time frame (usually a few weeks) and then refines and adapts this to changing requirements. Hence, XP trades risk against future effort: it reduces the risk that the team spends effort on things that may never be useful against additional development effort in case that the teams needs to implement a feature in the future that may have been anticipated today.

- *Direct communication is more effective than documentation*: Following XP, a project briefly documents user stories describing the requirements and specifying development tasks for the team. The documentation very short and serves more as a reminder to address an issue that as a detailed specification. Beside user stories, the primary documents to be created are source code: the implementation of the system as well as automated unit test drivers. In this sense, XP goes against standard software engineering practice by drastically limiting the amount of written documentation for a project. Nevertheless, anecdotal evidence shows that is works (e.g. [16]). The question is now: what makes it work? The answer lies in the combination of XP practices that focus on improving intra-team communication. First, XP uses small teams that share a workspace – improving informal communication (coffee breaks, overhearing discussions etc.). Second, a customer representative is co-located with the development team and, hence, resolving requirement issues happens fast. Third, pair programming makes sure that team members talk about their design and implementation in detail. Forth, shared code ownership leads to an overall understanding of the system design and implementation by many or all team members – if a new team member has a question, she can simply ask a colleague. Usually it is faster to explain a design face-to-face than describing it clear enough on paper (or in a CASE tool) so that others can understand. By reducing the amount of time spent on documentation of requirements and design, more effort can be spent on implementing additional software features for the customer – resulting in improved productivity of the team.[1]

Despite the fact that XP has some major advantages compared with more traditional software development methodologies, it 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 co-located.

*Scalability of the process*: Replacing documentation by communication seems to be a limiting factor for the scalability of the size of an XP team: If one design needs to be explained to n people and n gets larger, there will be a point when creating the docu-

[1] Productivity is here used in the informal sense of the number of functions provided to the customer divided by the time needed to develop them

mentation and letting them read it will take less time than explaining it n times in face-to-face communication.[2]

*Co-located teams*: XP requires all team members to be co-located and share the physical workspace. Co-location improves face-to-face communication and makes it easier to quickly ask questions. It enables collaboration and enhances coordination.

Overcoming the co-location requirement while preserving the high productivity and good coordination of XP processes is the focus of this paper.

In Section 2, we discuss some trends in the software industry that justify why thinking about distributed extreme programming is important. Section 3 gives an overview on our MILOS approach. Section 4 provides a usage scenario. We conclude with a summary and a look on future work.

## 2 Virtual Software Teams and Extreme Programming

While XP focuses on small, co-located teams, the 1990's saw the emergence of various kinds virtual software development organizations: teams of developers that worked on the same software but were distributed all over the world - using telecommunication and the Internet for communication, collaboration and coordination.

The same decade also saw a dramatic increase in freelance work [13], especially in the high tech field. As a result, Thomas Malone and his co-workers described the emergence of an e-lance economy [14] where freelance workers collaborate virtually over electronic networks to perform their jobs. E-lancing works best when all job related information and work results can be transferred over electronic networks. Software development is a prototypical example for this kind of work organization. This is demonstrated by new ventures like eLance (http://www.elance.com) or Asynchrony (http://www.asynchrony.com/welcome.jsp) that are building virtual marketplaces for e-lance work and whose focus is on software development and web design.

- The business advantage of virtual teams and e-lancing are primarily twofold:
- Increased flexibility in finding required resources when they are needed
- Reduced costs because of less training expenses and of outsourcing to markets providing cheaper labor

Examples of virtual software development teams are plenty. They reach from outsourcing small development tasks to an e-lancer to large-scale software development in the telecom or military industry where teams are distributed over the whole country or even the whole world. Other examples are open source projects like Linux or the Apache Web Server. These projects delivered complex software of high quality but the software development team never or, at least, very rarely met in the real world.

Open source projects share the XP focus on source code but they are neither co-located nor are the teams developing Linux or Apache small. This leads us to believe that the co-location requirement of XP is somewhat arbitrary and DXP seems to be possible.

---

[2] Even if one trainer explains the system requirements/design to many new developers at the same time (e.g. in a classroom setting), there will be a point where producing a document is more cost-effective than 1-to-1 or 1-to-many training sessions.

The term distributed XP was actually coined in [12]. They analyzed XP practices and show that only the Planning Game, Pair Programming, Continuous Integration, and On-Site Customer are influenced by the co-location requirement. They describe some initial experiences using of-the-shelf, not integrated tools (e.g. MS NetMeeting, CVS, e-mail) for communication, collaboration & coordination and reported several problems. Besides providing an integrated coordination approach, MILOS ASE deals with issues: handling of user stories and uniform access to a repository.

While Kirchner et al started from XP in their investigation of DXP, we started by analyzing how virtual software development teams currently work. We used open source projects as the base for the investigation. Open source projects are good examples of software development in a virtual environment. Analyzing their processes lets us determine

- How virtual software teams communicate, collaborate and coordinate their work
- How they share information and knowledge
- What tool support they use
- Where shortcomings are

Besides violating most standard software engineering practices regarding documentation of requirements and design, most open source projects demonstrate some common approaches how the development process is organized and how knowledge is transferred between heads of team members.

### 2.1 Open Source Development Processes

Open source projects need to ensure that relevant information is available to all participating developers. Here we need to differentiate between which information is seen as relevant (contents) and how it is made available to the developer community (modus).

All well-known open source projects maintain one or more Web site(s) as a focal point for information exchange. The Web site usually describes the mission and goals of the project and provides access to project related information.

Source code and documentation is usually accessible as one compressed file or via an Internet-based configuration management system (most often CVS).

Issue tracking systems, using a Web-based front end, are used to record bugs and for managing the workflow for fixing them. Usually, there are clear definitions when to submit a new bug report and how it will be handled.

Discussion groups or mailing lists are used to propose new features and extensions[3]. Newsgroups and mailing lists can also be used for getting help in case of problems with the open source software.

Most information is made available to the developer community in pull mode: developers are able to access the Web site of the source code tree whenever it fits into their schedule. The same holds for discussion groups. This lets the developer in control of his/her own time. Push mode is e-mail based. It is used for distributing bug reports, for discussing new features and for on-line support. Although e-mail is more intrusive than newsgroups, it still is an asynchronous communication medium.

---

[3] Sometimes new feature requests are stored in the issue tracking system.

Overall, open source development processes primarily use Internet technologies for supporting communication and collaboration. For example, the Apache Group states: "using the Internet and the Web to communicate, plan, and develop the server and its related documentation" [1]. A good example on collaboration support is provided by sourceForge (http://sourceforge.net/), a site that hosts many open-source projects.

## 3 The MILOS ASE Approach

The overall goal of the MILOS approach is to support project collaboration & coordination and organizational learning for virtual software development teams. 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". In this paper, we focus on how MILOS ASE supports DXP. [9] describes the knowledge management aspects in more detail.

### 3.1 Requirements on Tool Support for Virtual XP Teams

Using DXP 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 XP team to assign tasks to developers, set deadlines and get an overview on the current state of the project. Team members should be able to access their to-do lists and retrieve relevant information for performing their tasks easily.

***Synchronous communication***: 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 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].

### 3.2 Process Support in MILOS ASE

In this section, we give an overview on the support provided by MILOS to virtual XP teams. Background information can be found in [15]. In this paper, we cover only the DXP support aspect of MILOS.

MILOS supports the execution of globally distributed software development projects in several ways. Project managers are able to define tasks and decompose them into smaller subtasks. They can schedule them by using the Web-based user interface of MILOS. In addition, the information flow between tasks can be specified: For each task, a user is able to define expected outputs (e.g. the expected output of the "Develop work-

flow kernel" task is compressed library containing a set of Java source code files that is stored in the variable "workflow kernel"). These outputs can then be used as inputs for other tasks (e.g. the variable "workflow kernel" could be the input of the task "Develop workflow user interface"). In contrast to standard workflow engines, MILOS allows on-the-fly plan changes and redefinitions of the information flow, notifying team members affected by those changes and updating the workflow engine accordingly [15]

The MILOS framework nicely fits our requirements on DXP support. Nevertheless, we added the several extensions for supporting distributed XP:

- User stories: A new product type that represents user stories was added. In addition, whenever a new user story is entered, MILOS ASE automatically adds a task for implementing this story (see example in the next section) into the task list.
- Release and iteration planning: MILOS ASE allows easily defining and changing releases, iterations, user stories, and tasks. In a distributed setting, MILOS provides awareness on what is going on in the project based on 4 task levels from XP (release, iteration, user story, task).
- MS NetMeeting Integration: By integrating MS NetMeeting into our tool, we are able to support distributed pair programming and synchronous communication.

## 4 Using MILOS ASE for Distributed Extreme Programming

The following scenario illustrates the infrastructure provided by MILOS to support distributed extreme programming. Team members access the Internet with a Web browser and connect to the MILOS ASE server (http://sern.ucalgary.ca/~milos). 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.

### 4.1 Creating User Stories

After creating a project and assigning a project manager to it, the customer is able to enter story cards into the MILOS system (see Figure 1). The top part of shows a menu that allows accessing all components of MILOS (Tasks, Team, Experience Base, etc). The left side shows a context-dependent navigation menu that enables to create a new story card. When a programmer pair starts working on a user story, it will contact the customer and discuss the story with him and update the description of the story if needed.

When a user creates a new story, the MILOS ASE system automatically creates a task to handle this story and places it under "Unassigned tasks". These unassigned tasks can then later be assigned to a specific iteration.

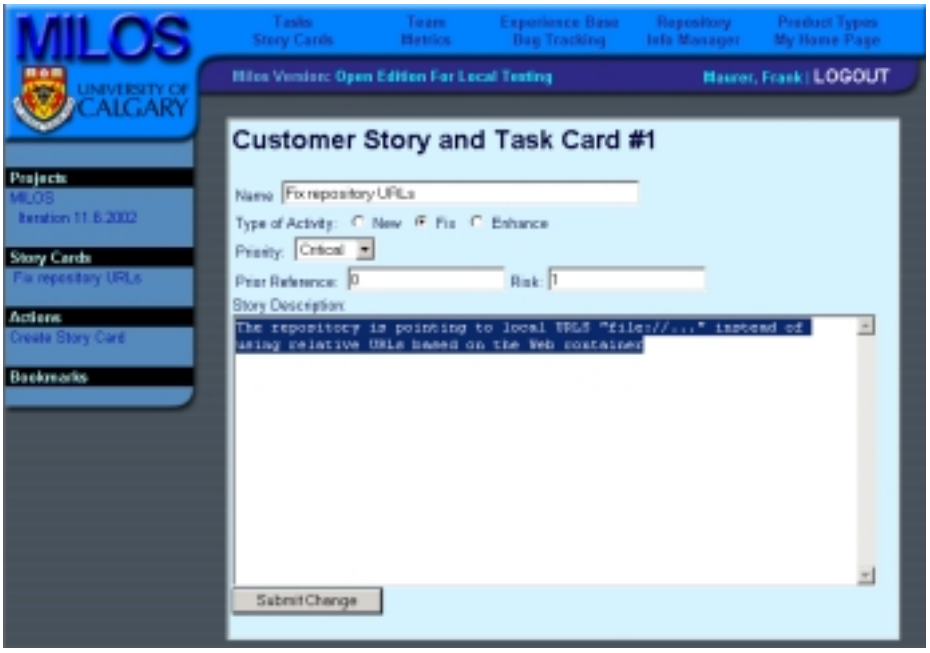


**Fig. 1.** Story card

Using the release and iteration planning user interface (see Figure 2), the developers may split a user story into several tasks as well as assign them to an iteration. Using the up/down links shown in the UI, a developer or customer may easily move user stories between iterations (and iterations between releases).


**Fig. 2.** Release and iteration planning user interface

After having decomposed user stories into concrete development task, the developer may describe the task in more detail. Tasks are part of a specific project and can be accepted by one team member. For each task, the manager may enter planned start and end dates. In addition, the users are able to define the inputs and outputs of processes. 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. 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 project manager may keep a close eye on the progress of each task by watching "percentage complete" values and steer the team in the correct direction if the requirement for the next build will not be met. We believe that this information is useful in a distributed setting as the informal information flow between team members is more difficult than in a co-located environment. As a result, a little bit more information needs to be "put on paper" than for co-located teams.

### 4.2 Pair Programming

MILOS keeps track on whose team members are currently logged in. A developer is able to pair up with one of them using the application sharing and audio/video capabilities of MS NetMeeting. Initial tests done by two of the MILOS team members[4] indicate that state of the art networks are sufficiently fast to support distributed pair programming. They also said that a video link is not useful as long as the pair know each other well. They also recommend that both machines should use the same screen resolution (to avoid scrolling).

## 6 Related Work

Related work comes mainly from two areas: Software Process Support and (Distributed) Extreme Programming. As we already discussed XP and DXP in Sections 0 and 0, 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 [6].

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.

---

[4] Darryl Gates & Sebastien Martel paired up for a couple of hours over the Internet while developing MILOS source code.

Software process modeling and enactment is one of the main areas in software engineering research. Several frameworks have been developed (e.g. procedural [17], rule-based [11][18], 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 supports communication, collaboration and coordination of DXP teams.

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

Our future work will focus on four aspects:

- Formal evaluation of the approach
- Extreme federations
- Knowledge management for DXP

*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 co-location 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 thirty 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 need 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 focuses on verbal communication for knowledge exchange. That 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 runs 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.

## References


[1] N.N.: About the Apache HTTP Server Project, http://httpd.apache.org/ABOUT_APACHE.html, 1999 (last visited March 2002)

[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] Kent Beck: 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, 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).

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

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

[8] P.K. Garg, M. Jazayeri: "Process-centered Software Engineering Environments". IEEE Cumputer Society Press, 1996.

[9] Harald Holz, Arne Könnecker, Frank Maurer: Task-Specific Knowledge Management in a Process-Centred SEE, Proceedings of the Workshop on Learning Software Organizations LSO-2001, Springer, 2001.

[10] Ron Jeffries, Ann Anderson, Chet Hendrickson: Extreme Programming Installed, Addison-Wesley Pub Co, 2000, ISBN: 0201708426

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

[12] Michael Kircher, Prashant Jain, Angelo Corsaro, David Levine: Distributed eXtreme Programming, Proceedings XP-2001, Villasimius, Italy, http://www.xp2001.org/program.html (last visited July 2001)

[13] Laubacher, Robert J., Malone, Thomas W.: Flexible Work Arrangements and 21st Century Worker's Guilds, Initiative on Inventing the Organizations of the 21st Century, Working Paper #004, Sloan School of Management, Massachusetts Institute of Technology, October 1997, http://ccs.mit.edu/21c/21CWP004.html

[14] Malone, T. W., Laubacher, R. J.: The Dawn of the E-Lance Economy, Harvard Business review, Sep-Oct 1998.

[15] 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.

[16] James Newkirk, Robert C. Martin: Extreme Programming in Practice, Addisson-Wesley, 2001, ISBN: 0-201-70937-6

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

[18] 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).

[19] Roger S. Pressman: Software Engineering: A Practitioner's Approach, Fourth Edition, 1996, ISBN: 0-07-052182-4.