Working Group Report on Coordinating Distributed Software Development Projects

Harald Holz*, Sigrid Goldmann* and Frank Maurer#

*University of Kaiserslautern, AG Expert Systems, P.O. Box 3049, 67653 Kaiserslautern, Germany
e-mail: {holz, sigig}@informatik.uni-kl.de

#University of Calgary, Department of Computer Science, Calgary, Alberta, T2N 1N4, Canada
e-mail: maurer@cpsc.ucalgary.ca

Abstract

This paper summarizes the work presented at the WET ICE '98 workshop on "Coordinating Distributed Software Development Projects" as well as the ensuing discussions that arose in the course of the workshop.

1 Introduction

Compared to other engineering disciplines, software engineering (SE) still is a relatively young field. Although in recent years progress has been made in software development methods, techniques, and tools, software development is still more an art than an engineering discipline and basic problems remain:

- software development is expensive and time consuming
- software projects often run out of time and over budget.

The success of software projects still relies heavily on the competence of the individual developers involved and not on the maturity of the organization.

Another problem stems from the fact that in software engineering special emphasize lies on design tasks whose results are highly volatile even in established engineering disciplines. In practice, even the implementation or coding phase in software development resembles more a design activity than a typical production phase in other engineering disciplines: Code can be changed easily whereas e.g. changing a house that has been built is quite an effort. As a consequence, the design and code of a software product almost always is subject to changes, whereas other engineering disciplines have a design phase followed by a lengthy production phase.

In effect, the efficient management of changes in requirement, design, and code documents appears to be one of the main problems of software development. Questions that arise are

- how can one handle the effects of changes to software artifacts
- how can we constrain the effect one change has to other parts of the software system
- how can we ensure proper flow of change information.

Techniques for change management have to be provided in order to ensure the timely conclusion of a project with high-quality products.

Another problem that arises in software engineering is the fact that there exist a large number of software development tools and data formats, that need to be integrated into a more defined software development process as projects grow. Middleware tools are needed to facilitate this integration, allowing legacy tools to communicate with each other so that existing tools can continue to be used, and existing data need not be discarded as software engineering technology changes.

The above points emphasize the necessity for software project coordination and communication support even in ordinary, local software development projects. Lately, additional difficulties have arisen with the need to globally distribute software development projects. For economical or technological reasons, projects cannot always be restricted to one company or a single location:

- Time to market needs to be reduced in the Internet-age: Business success is often strongly determined by being the first selling a new technology.
- Some software projects grow too large to be handled in a single location.
- Knowledge and skills needed in a project may not be available locally but may be distributed all over the world.
- Outsourcing development activities to emerging market countries may provide the competitive edge to get a contract.

Such global distribution of software projects drastically increases the need for coordination and communication support. Additionally, it adds another dimension to the problem of coordinating such a project: Not only does the project's complexity make it hard for people to figure out whom to contact when a questions comes up. In a distributed project they cannot just walk over to the person in question. Even picking up the phone and calling the other person might not be feasible if the project is distributed over different time zones. Discussions in the style that people are used to, i.e. official group meetings, or informal discussions over some idea on a piece of paper, become impossible. Instead, the necessity arises to support asynchronous as well as synchronous collaboration with computer tools.

To summarize, three main research topics form the field of computer-supported cooperative work were frequently referred to in the workshop: communication, collaboration, and coordination. Techniques developed in those areas need to be adapted, extended and integrated with state-of-the-art SE tools to support distributed software development activities. All twelve workshop presentations dealt with one or more of these topics.

In section 2 of this paper, we further discuss these three research areas and briefly summarize the papers presented on each topic. Section 3 sums up the workshop's discussions on the applicability of existing technologies and their shortcomings. Section 4 concludes this report.

2 Presentations

2.1 Coordination

Project coordination can be interpreted as the attempt to get the **right information** to the **right people** at the **right time**. In other words, this research area is concerned with the trade-off between minimizing information overload versus avoiding a lack of information: sending information to people who are not interested in it (thereby making it hard for them to filter out the facts that are important to their work) is as bad as having people not being informed of facts that are important to them.

In order to address these issues, process modeling and enactment support has been proposed: process modeling provides a descriptive representation of the process to be carried out. Process models contain implicit and explicit dependencies which then allow for automatic notification support when changes occur during process execution. Several presentations at the workshop dealt with process modeling and enactment:

K. Alho [1] introduced a common dominator representation (CDR) for process modeling and enactment. The CDR allows loose integration of different process, product and collaboration platforms. This avoids the cost of re-modeling when different organizations that take part in a virtual software corporation use different modeling languages and technologies.

P. Benjamin [3] described the architecture of a framework that supports the whole project life cycle, consisting of project definition, process design, analysis, enactment, monitoring and redesign. Process design is supported by reusing templates from a library while redesign is triggered by proactive and reactive sentinels that monitor process execution.

G. Faustmann [4] presented a petri net-based process modeling and enactment approach that allows the refinement of workflows during enactment. Refinements are chosen from a set of predefined alternatives and can be marked to specify various forms of enforcement. In combination with an exception handling mechanism, situation-dependent coordination can be accomplished.

S. Goldmann [2] and *F. Maurer* [11] introduced a flexible process modeling and enactment environment that focuses on managing the consequences of changes during enactment. Coordination activities are explicitly represented and triggered at appropriate times. Their system integrates an industrial tool for project planning and scheduling. The process engine is able to react to plan changes.

R. Foley [6] presented a formalism to support the development process within a virtual software corporation (VSC). The formalism allows the representation of the more complex nature of the communicational aspects between individual process actors of geographically dispersed teams and organizations. The approach provides a solution for representing a network of commitments (instead of a management hierarchy) to be found in VSCs. He illustrated his work with a real-world example of a VSC that was distributed over three locations: Management, design teams as well as core functionality implementation were located in London and Edinburgh, whereas user-interface implementation was outsourced to a company in Singapore.

J. Grundy's presentation [5] dealt with the integration of process modeling/enactment environments and tools for collaborative editing. This allows changes to artifacts to be categorized according to the process steps in which they occurred. A collaboration component can be "plugged in" into third-party tools to provide both synchronous and asynchronous editing capabilities. Their system provides a change notification mechanism that can be configured by developers via a visual language.

In the ensuing discussions on process modeling and coordination support, a number of requirements for a process support tool were identified: Since VSCs are becoming more common in software development projects, process models should not only be definable for single-site development, but also for a distributed project. In a VSC, each individual company should be able to model and enact their own process, using their modeling language of choice as well as their own enactment environment (leading to the requirement that different enactment engines need to be able to exchange information). Process modeling tools should allow for bottom-up process definition (and support later integration of the sub-processes), as well as provide a top-down way of defining processes. The resulting process model should reflect a network of commitments, i.e. the direct responsibilities and dependencies between tasks.

Since the process, as well as the product, in software development is subject to frequent changes, process engines need to support changes "on the fly", i.e. remodelling must be allowed during enactment, and the process engine must be able to react to such changes in two ways: First, it needs to be able to update its internal state to reflect the current state of the software development process. Second, it needs to be able to notify people involved in the process of the changes that concern them. In other words, a process model should realistically capture who should be informed of what change, in order to avoid information overload of team members.

Another aspect of coordination is ensuring data consistency, as well as configuration management of products generated during the software development process. The participants agreed that these questions are a major aspect of coordinating distributed software engineering projects, and that substantial work is still needed on the topic.

2.2 Collaboration

Collaboration support addresses the problem of "joint work" between participants that cannot be physically present at a common location. Synchronous collaboration, e.g. formal meetings and ad hoc discussions, need to be supported, as well as asynchronous collaboration, e.g. a "white board" that is waiting in a virtual "room" for people to come in and leave a note for everyone to see. Another aspect of collaboration are virtual workspaces and collaborative editing of artifacts that result from a development process.

D. Herlea [8] reported on the use of an industrial groupware tool (TeamWave) for the requirements engineering (RE) phase. The tool has been tailored to support collaborative negotiation of requirements between geographically separated agents. The RE process can be structured via customized virtual rooms while the discussion of multiple perspectives is facilitated by specific TeamWave tools.

M. Penedo [12] presented an industrial experience report on introducing collaborative infrastructure technologies into a companies working practises. Synchronous data conferencing could successfully be used for impromptu meetings, accessing remote experts, collaboration in design or training. Concepts for the integration of synchronous and asynchronous capabilities, like virtual rooms, have been found to be important but too immature.

The discussion on synchronous and asynchronous collaboration support showed that collaborative spaces, and the management of the information flow inside and between such spaces require further research. The question how to integrate traditional software engineering support tools with groupware approaches and collaborative spaces is still an open issue. Two alternatives are

- to extend traditional software engineering support tools by groupware technology, or
- to use existing groupware tools to support software engineering activities.

Another way to deal with collaboration in distributed projects is to try and organize the task at hand in a way that few or no interdependencies exist between the subtasks, which can then be worked on independently.

G. Matos [10] presented a formal approach for independent component development with the aim of avoiding communication between developers of different components. While data dependencies are defined early in the project, control interactions are handled during system integration. To correct faulty component interaction, the approach allows the automatic synchronization of component behaviour, making it consistent with specified receptive safety rules.

2.3 Communication

The topic of communication addresses the need to integrate different legacy tools and data formats into a network in which messages can be sent and data exchanged between the tools.

S. Dossick [9] presented a middleware framework for artifact storage in groupware environments. The system references external objects that can be accessed via appropriate protocols written for the framework; additional protocols can be integrated easily. Furthermore, an initial set of services like persistency, transaction management and workflow functionality are provided by the framework.

M. Hao's presentation of SmallSync is a middleware system for the integration of different diagnostic and visualization tools for monitoring distributed Unix processes [7].

An issue that was raised during the discussion on the topic of communication support was the fact that middleware is not only needed to support the integration of data and legacy tools (like text editors, compilers, etc.) used for the generation of products in the software development process, but we also need to support enactment of distributed processes modeled at the different sites in a virtual software company.

Middleware is needed to loosely integrate different process engines used to enact different sub-processes. It provides a communication mechanism for these process engines to exchange information about the state of the different sub-processes.

3 Technology for Software Development Support

In order to deal with communication, collaboration, and coordination issues of software development, one or more of several existing technological approaches can be used: Groupware tools provide support for collaborative work, workflow management approaches support the coordination of SE processes, change impact analysis and change management techniques can be used to send notifications to an appropriate set of users, middleware and standards exist to integrate diverse data formats and provide interoperability between legacy tools. Agent technology introduced recently allows for active artifacts to communicate with each other and with human users. Data stucturing approaches like XML provide us with a standard means to exchange products generated in the software development process.

However, none of these technologies by themselves are enough to solve the problems posed by distributed software engineering. In order to provide useful support for that complicated process, methods and tools must integrate coordination, collaboration *and* communication support, i.e. it must integrate, to a degree, all of the above technologies, as well as provide support for project monitoring, and configuration management. To which degree each of the above technologies can be helpful in supporting distributed software engineering, and what other technology might be needed in order to provide a really satisfying support tool, is still a question that requires substantial work.

4 Conclusions

Even in traditional software engineering, there are still problems to be solved concerning the management and coordination of complex development processes. Distributed software development and virtual software companies cause additional difficulties and challenges that need to be addressed. One reason for this is the additional communicational (both human and technical) complexity caused by global distribution of a development project. If a project is distributed over different companies or even countries, organizational cultures are sure to diverge, thereby causing misunderstandings, and necessitating a process support that allows for different process structures, as well as for different control strategies (such as strict enforcement of activities, guidance, and free choice of the activities to work on), all in a single distributed process. Different modes of communication are necessary in a globally distributed process: Synchronous communication (e.g. meetings and discussions) needs to be supported, as well as asynchronous communication (providing passive and active access to information).

Interoperability is an issue that needs to be addressed at several levels: different data formats need to be exchanged between different tools, different process engines need to communicate with each other, and, last but not least, people from different (organizational and general) cultures need to talk to and understand each other.

Another topic that is closely related but was only briefly mentioned in the workshop, and should be addressed in the future, is access control and ensuring privacy for the companies involved as well as the individual developers working in a VSC. Any tool not dealing with these questions would not be easily accepted in practice: A tool that does not ensure the user's privacy, may not be used by software developers.

This workshop mainly addressed technical problems. Other topics, such as a tool's acceptance in terms of usability and adaptability, were mentioned, but not dealt with in detail. Social and cultural issues, such as organizational and geographical differences in culture, were not discussed, although the participants agreed that they are at least as important as the technical questions.

Literature

[1] K. Alho, R. Sulonen: *Supporting Virtual Software Projects on the Web*, in: [13].

[2] F. Bendeck, S. Goldmann, H. Holz, B. Kötting: *Coordinating Management Activities in Distributed Software Development Projects*, in: [13].

[3] P. Benjamin, M. Erraguntla, R. Mayer, M. Painter, Ch. Marshall: *A Framework for Adaptive Process Modeling and Execution (FAME)*, in: [13].

[4] G. Faustmann: *Flexible Handling of Work Processes by Situation-dependent Support Strategies*, in: [13].

[5] J. Grundy, J. Hosking, R. Mugridge: *Coordinating Distributed Software Development Projects with Integrated Process Modeling and Enactment Environments*, in: [13].

[6] Z. Haag, R. Foley, J. Newman: *A Deontic Formalism for Co-ordinating Software Development in Virtual Software Corporations*, in: [13].

[7] M. C. Hao, D. Glajchen, J. S. Sventek: SmallSync: *A Methodology for Diagnosis & Visualization of Distributed Processes on the Web*, presentation only.

[8] D. Herlea, S. Greenberg: *Using a Groupware Space for Distributed Requirements Engineering*, in: [13].

[9] G. E. Kaiser, S. E. Dossick: *Workgroup Middleware for Distributed Projects*, in: [13].

[10] G. Matos, J. Purtilo, E. White: *Automated Enforcement of Receptive Safety Properties in Distributed Design*, presentation only.

[11] F. Maurer, B. Dellen: *An Internet Based Software Process Management Environment*, in: [13].

[12] M. H. Penedo: *Experimenting with technology associated with Mobile Desktops*, in: [13].

[13] Proceedings of WET ICE 98, IEEE Press, 1998.