ORGANIZATIONAL ASPECTS OF DISTRIBUTED SOFTWARE DEVELOPMENT

Josef Altmann [1]), Heinz Dobler [2])

Abstract

Software production has become an industrial task usually involving teams of programmers working on complex problems to produce large, even huge software systems. A growing share of all software development work is being done by globally distributed teams. The management of software engineering teamwork, especially of a temporally and/or spacially distributed team, presents an enormous organizational challenge as well as an intricate technical problem, as such distributed teamwork requires tool support for coordination of cooperative activities, maintenance of project control, and sharing of information. We present an overview of the managerial (mainly organizational) basics and aspects of distributed software development, give an overview of the potential and the limits of some of the published research projects, extract the design principles for the construction of cooperative software development environments, and formulate a model for cooperative work processes in software projects.

1. Introduction and Motivation

The complexity of real-world problems that we must tackle is steadily increasing. While the dramatic improvements in computer hardware and in low-level system software technology provide the raw computing power necessary to handle these problems, we continue to wait for the software solutions to our problems. Why?

Software engineering, i.e., software development "in the large", has never been a simple task. Programming "in the small" is fun and relatively easy to learn: Following the examples in a good textbook soon can provide the skill to generate new programs from the given patterns. However, this method of solving simple problems with pattern matching augmented by intuition cannot be extrapolated to large software systems. Furthermore, the approach of starting with small programs and modifying and/or extending them until they meet the functional requirements does not lead to any success either.

[1] Christian Doppler Laboratory for Software Engineering, Johannes Kepler University, A-4040 Linz, Austria

[2] Upper Austrian Polytechnic University for Software Engineering, Hauptstraße 99, A-4232 Hagenberg, Austria

Over the last two decades, thousands of software-centered projects ended up with missed deadlines, blown budgets and flawed products [5]. The situation nowadays is even more critical. Mastering large software development projects has grown more complex, not only because project sizes have increased, but also because the software engineering team is increasingly distributed across time and/or space. Besides all the difficulties that traditional software engineering management has to contend with, the nature of distribution brings additional problems that we address in the following chapters: coordinating team activities and maintaining project control. We do not discuss organizational structures or software process models but focus on the prerequisites for cooperative software development in a distributed team and define the necessary tool support via an abstract model.

2. Organizational Aspects

The organizational provisions in the area of software development have to support *communication*, *coordination* and *cooperation* during the development process in order to continuously improve the quality of the process and the product.

Communication forms the basis for any kind of cooperation. It is defined as the technology and process of transportation and interchange of information. Most of the communication tools (e.g., electronic mailing systems) in use today support asynchronous communication only. Modern synchronous tools (e.g., video conferencing systems) promise to improve the quality and efficiency of information interchange.

Coordination is based upon adequate communication technology and processes. It comprises all the activities necessary to synchronize the parts of the divided tasks within the context of a larger work process. For effective coordination, asynchronous and synchronous tools have to support the use (e.g., reading,, compilation) and the editing of common electronic documents as well as the interchange of information concerning common tasks. According to Burger [6] the process of project coordination consists of splitting a task into subtasks, allocating subtasks to team members, ordering the subtasks over time (e.g., sequencing and synchronizing), and combining/integrating the results of the subtasks. Within a well-coordinated project, the team members work on their subtasks and strive to reach their individual subgoals. Only the combination/integration phase delivers results that meet the overall goal.

Cooperation is a special kind of coordination among the team members; it is necessary for defining common goals and for achieving common work results. This definition makes it clear that

cooperative work implies synchronization and coordination during the production process. Borrowing from Bauknecht et al. [3], we define the requirements of cooperation as follows:

- *Identification of goals*: the cooperating partners have to agree in common goals.
- *Compatibility of plans*: in addition to the identification of goals, the plans of two or more cooperating partners have to be synchronized.
- *Resource exchange*: the exchange of resources and work on a common work piece are prerequisites for cooperation.
- *Regulation*: compatibility of plans and resource exchange often cannot be completely defined in advance; cooperation requires flexible negotiation and continuous adaptation.
- *Control*: the possibility of controlling the activities of the cooperating partners is a necessity in order to enable common assessment of progress.

While the definition and the elements of cooperation as stated above are disputed, there is general agreement that cooperation is a mutual influence on distributed tasks or work plans.

A further investigation of cooperation [10, 17] resulted in different *forms* and several *dimensions* of cooperation. According to the amount of freedom and possibilities of interaction, two *forms of cooperation* can be distinguished [10]: (1) *team-based* (flexible arrangement of structure and course) and (2) *structured* (technical aspects influence cooperation, strictly defined course) cooperation. Other characteristics of cooperation can be defined via orthogonal dimensions [17]: (1) *bilateral* versus *multilateral* cooperation (number of partners involved), (2) *conjunctive* versus *disjunctive* cooperation (means of goal achievement), and (3) *direct* via *indirect* cooperation (time and location of partners).

Traditional cooperation approaches are product-oriented and arrange the software development process in sequential phases. Subproducts like documents and prototypes define the interfaces between these phases. Research (e.g., Dourish [8] as well as Pomberger and Blaschek [18]) has shown that product orientation is inadequate for complex software projects: the combination of static descriptions with an idealistic sequential development process neglects creativity and cooperative aspects of software engineering.

New approaches, some of them adopted from other disciplines like computer integrated manufacturing (CIM), try to consider mutual dependencies and cooperative recognition. Communication, coordination and cooperation represent the basis of these approaches: *participatory design* (integration of users and mutual learning), *concurrent engineering* (no sequential phases but parallelism wherever possible to stress mutual dependencies), and *total quality management* (involving all staff members, concentrating on the quality of the process as well as the product, and giving consideration to mutual dependencies). Besides the concrete approach taken, cooperation is an essential part of software development; over time the form and

the dimension of cooperation may change and cause cooperation conflicts. Therefore any support of the software development process has to focus on ensuring efficient cooperation.

Having identified the technical prerequisites for teamwork (communication, coordination and cooperation), in the following we investigate different factors influencing teamwork. Teamwork can be characterized according to some very general attributes. Although the various aspects of software development are heavily influenced by the underlying process model, in this paper we concentrate—without any investigation of these process models—only on the characteristics of teamwork (e.g., Hruschka [13], Kraut et al. [14] as well as Pomberger and Blaschek [18]):

- *Size and complexity*: Increasing size and complexity as well as the necessity for steady adaptation result in the need for cooperation on the part of several persons.
- *Creativity*: Software development is an intellectual activity. The problem solving process is based on creativity, and the results are not predefined. This demands flexibility and permanent learning.
- *Uncertainty*: Planning and execution of software development projects according to incomplete or changing requirements is bound to uncertainty.
- *Informal communication*: While process models define formal channels of communication, several investigations (e.g., Bischofberger et al. [4]) stress the necessity and usefulness of spontaneous and flexible informal communication.
- *Standardized subtasks*: Software development includes non-application-specific activities (e.g., code reviews). Predefined procedures assist in the execution of such tasks.
- *Process documentation*: Pomberger and Blaschek [18] stress the importance of documenting the software development process itself, e.g., to allow later reconstruction of any design decisions.
- *Long transactions*: Some processes need a long of time to execute and may block common resources, thereby influencing other tasks.
- *Software bureaucracy*: Large teams working on complex systems often have to obey strict guidelines and rules, often implemented within CASE tools.

Empirical results of an investigation of teamwork in 29 software projects in 19 corporations showed that 40 % of the total development effort is spent for coordinating team members [12]. Thus the necessity of efficient support of cooperation is obvious.

The situation worsens when we consider that large software systems cannot be built in a monolithic form. The systems have to be broken down into subsystems. This implies the necessity of further splitting the cooperative software development process according to functionality, time and location, resulting in distributed software development.

Based on the general term cooperation, we define *cooperative software development* as comprising all communication and coordination activities undertaken within a software project in order to plan, execute and synchronize functionally, temporally and/or spacially distributed processes as well as

process- and product-specific activities of all team members working toward the common goal of constructing a software system.

3. Overview of Research Projects and Tools

Besides organizational and social support, efficient distributed software development needs specific tools. In the broad area of computer supported cooperative work (CSCW), many different systems already have been constructed to assist in teamwork. The following kinds of systems can be distinguished:

- *Communication systems* provide support for asynchronous and synchronous communication, e.g., mailing systems, news groups, chatting and conferencing systems).
- *Coordination systems* support the planning and synchronization of cooperative activities, e.g., workgroup calendars, electronic workspaces for document sharing and exchange.
- *Information administration systems* support the construction, administration and use of process and product documentation, e.g., muti-user editors and hypertext systems.

These systems proved to be unsatisfactory for cooperative software development, especially because of their insufficient conceptual and technical integration of process and product views. Research projects dealing with the problem of coordination and control of a team of software engineers can be found in several areas:

- Research on computer-supported cooperative work (CSCW) does not focus on software production, but investigates all activities carried out by a team of persons on electronic documents of any kind using computers (e.g., workflow management in general business processes). Whereas CSCW efforts, especially the collaborative workflow approach, potentially influence cooperative software development, the corresponding tools only support a few of the features necessary for cooperative software development, e.g., synchronous editing [7].
- The process-centered software engineering (PCSE) approach (see, e.g., Osterweil [16], Lehmann [15], and Ghezzi et al. [11]) tries to automate the software development process according to a more or less specific software engineering process model (e.g., the traditional ones like the waterfall model and the spiral model, or the modern ones like the prototyping model and the object-oriented model) but neglects important human characteristics of software development such as creativity, uncertainty and informal communication [14].
- Cooperative software engineering (CSE), a recent branch of the broad software engineering (SE) research activities, offers promising approaches. Systems in this category of policy-driven cooperation (such as conventional version and configuration management) concentrate on formal as well as informal cooperation through the connection of structured and unstructured information (e.g., annotations) to documents (cf. Bischofberger et al [4]).

The goal is to implement a suitable development environment based on design principles we formulate in the next chapter. In addition, existing development environments can be evaluated according to these design principles.

4. Design Principles for Development Environments

The introduction and use of CSCW systems in general implies changes and adaptations in the structure of an organization and in the performance of common work. These modifications cannot be anticipated completely in advance. For many authors (e.g., Piepenburg [17]) this represents one of the main problems in the development of the groupware software. To grasp the overall dynamics of cooperative structures and processes, only an iterative and evolutionary approach—called *cycle of design* in Dourish [8]—can be used. The following design principles serve the purpose of providing a basic understanding of distributed cooperative software development processes and activities to support:

- *Transparency*: The term transparency can be defined in a software engineering or an application domain specific context. In the former, transparency is closely linked to the development of distributed systems: the distribution of the components is hidden during development. With respect to the application domain, transparency involves the representation of the distribution of work pieces and activities for users of groupware tools. The latter view is the important one in the context of distributed software development—in contrast to development of distributed software. In the context of cooperative distributed software development, transparency means that the team members know the location of the software artifacts, the distribution of the development activities and their situation of concurrency to the other members of the team.
- *Individual and group-oriented view of the work process*: Activities are split up and allocated to different workgroups and/or team members. According to the distribution of tasks, the team members need an individual view of the development process to gain an overview of their responsibilities. In addition, a group-oriented view is necessary in order to judge the activities in the context of the team and to coordinate and synchronize the activities.
- *Common workspace*: The purpose of a common workspace is the preparation and consistent administration of process- and product-specific electronic documents within the software development project. The members of a common workspace need an environment for cooperation organization, structuring and performance of their work. Organization comprises the definition of the members as well as the work pieces and tools they use. Team members who assume organizational roles are granted access authorization and have defined duties.
- *Cooperative awareness*: Orientation in cooperative software development processes is one of the key factors for successful teamwork. A prerequisite for cooperative awareness is that the team members have the knowledge and a deeper understanding of the structure of the

development process. This requires an information exchange system providing the team members with data on ongoing and finished activities in order to let them know about the activities of the cooperation partners.

- *Negotiation as a possibility for the resolution of conflicts*: Conflicts should not and in reality cannot be completely regulated using formal rules and/or automatic procedures. Therefore the cooperating team members should be granted total control over the cooperative work process and be allowed to responsibly utilize their freedom as far as possible. The concrete amount of cooperation is thus negotiable among the team members according to the situation or problem.
- *Availability of predefined processes*: The standard sequence and dependencies of specific development activities in cooperative development projects should be predefined in the form of plans, instructions and process patterns. The main objective of the availability of predefined work plans lies in the support they provide, not in the strict regulation as typically imposed by workflow management systems. Experience demonstrates that complete and executable process definition is impossible for distributed software development.
- *Availability of predefined documents*: Document templates defined in advance (e.g., for error reports and change requests) provide a common framework for efficient and uniform recording and structuring of process- and product-specific facts.
- *Communication across time and location*: The temporal and/or geographic distribution of a team implies further coordination problems apart from division of labor. Tools allowing asynchronous and synchronous information exchange via informal communication (e.g., questions and answers, proposals, hints) have to be supported conceptually as well as technologically.
- *Implicit and explicit information exchange*: Explicit information exchange takes the form of explicit actions of the team members, who send and receive messages, reports and annotations via electronic mail or blackboard systems. Implicit information exchange occurs when update operations on common artifacts automatically result in tools sending events or signals to other tools or user interfaces, thereby informing the cooperation partners of any changes.
- *Information on the process history*: The execution of the development process implicitly defines its history, which has to be documented and made available for later access by the cooperation partners. Process history should contain information on the executed activities, disturbances, problems, unforeseen dependencies between activities, alternative solutions, insights, etc. Such information increases the overall understanding and provides efficient problem solving in later phases of the project via reuse of experiences. For maintainability and reusability, the process history gains even more importance.
- *Status and context information*: The necessary status information is either of organizational or technical nature. Organizational information comprises, e.g., team structure and the allocation of responsibilities, whereas technical information concentrates on the status of activities, reasons for modifications, and hints for later extensions.

- *Suitability of information*: The possibility for team members to complement any software artifacts (e.g., design documents, source files, prototypes) with context and additional information (e.g., comments, links to other sources of information) supports cooperative work.
- *Document and configuration management*: The huge amount of artifacts (different versions in the form of variants and revisions) produced during large software development projects has to be managed in a space-efficient, consistent and comprehensive manner.

Distributed development of object-oriented software requires additional adaptations, because of, e.g., the incremental design and implementation of classes as well as the identification of classes for potential reuse; increased communication and cooperation between workgroups is needed.

The design principles listed above cannot serve as concrete guidelines for the implementation of a tool (set) but serve to visualize the requirements for a distributed cooperative software development environment. The main advantages that a development environment based upon the design principles delivers are increased productivity, improvement of understanding, better quality of results (e.g., less errors, higher maintainability), and support for project management. In the next chapter, we formulate a model that can serve as a starting point for tool development as described in Altmann and Weinreich [1] and Altmann [2].

5. Model for Cooperative Software Development

The requirements formulated above form the basis for a lean conceptual logical framework, a model for a cooperative software development environment (CSDE) supporting temporal and/or spacial distribution of team members.

The central notion of the model is that of a *work package*. The idea is to split up a complex activity in sub-activities that are assigned to team members. Hence cooperative activities are organized as a hierarchy of workpackages that describe isolated activities executable by a single person. Workpackages have a responsible person, a performer, a deadline, a specification, other participants, necessary artifacts, and possibly subtasks. The workpackage specification describes the goal of the activity and helps to provide an overall understanding. It may contain attached artifacts such as documents, annotations and messages, as well as links to additional sources and tools.

A number of related workpackages form a *workcontext*. Product-centric or process-centric aspects can be taken into consideration when defining such a workcontext. At the highest level of abstraction a *workspace* groups several workcontexts that are relevant to achieve the common goals of a workgroup. Workspaces contain additional project-specific information about persons and

groups, document templates, guidelines, predefined work procedures and time schedules. A predefined work procedure consists of a sequence of work steps or subtasks that represent a guideline for how to perform a standard or routine activity.
Concurrency control mechanisms with rigid locking and conservative serialization have certain drawbacks in asynchronous collaboration. Following the idea that CSCW applications should inform rather than constrain and should provide a medium that can be adapted to suit the flexible nature of the work of the users, the model is supplemented with a flexible cooperation model. Some conflicting situations need to be solved by negotiation among the involved team members. Therefore the cooperation model distinguishes between various levels of participation and competence. All workspace members have unrestricted read permission for all work packages contained in the workspace. A set of people involved in a workspace, the *controllers*, are constructors and owners of a package object. Controllers are responsible for the completion and result of a workpackage; therefore they have read/write access to the package's attributes. In addition, by granting or revoking corresponding access rights they can allow or refuse another user, the *executor*, to perform work. This kind of cooperation allows the users to know exactly who can do what with a certain workpackage. The controller and the executor of a workpackage negotiate and coordinate their work on a package using informal communication, such as electronic mail, annotations, phone, or face-to-face.

This clear exposition of the model allows explicit conflict resolution due to simultaneous work on some objects. The described cooperation model further provides a certain kind of awareness within the development process but does not enforce a strict locking model.

6. Conclusion and Further Work

We have presented an overview of the organizational basics of distributed software development, discussing communication, cooperation and coordination as prerequisites for teamwork in general. After an overview of research projects, sketching their potential and their limits, we extracted the design principles for the construction of cooperative software development environments and formulated a model for cooperative work processes in software projects.

We have sketched this model in a previous paper [1]; however, the purpose there was to form the basis for the description of a tool focusing on process-oriented activities, leaving control to the persons involved in cooperative work. In another paper [2] we describe the design principles, the model and the object-oriented cooperative software development environment in detail. Experience with the tool shows that a number of limitations remain to be investigated (e.g., lack of sophisticated views of workspace structures, deficiencies in information filtering).

7. References

[1] ALTMANN, J., WEINREICH, R., An Environment for Cooperative Software Development Realization and Implications; in: Proceedings of the 31st Hawaii International Conference on System Sciences (HICSS), Wailea, Hawaii, USA, Jan. 6 - 9, 1998.

[2] ALTMANN, J., Kooperative Softwareentwicklung: Rechnergestützte Koordination und Kooperation in Softwareprojekten, Ph.D. Thesis, Johannes Kepler University, Linz, Mai 1998.

[3] BAUKNECHT, K., MÜHLHERR, T., SAUTER, C., TEUFEL, S., Computerunterstützung für die Gruppenarbeit, Addison-Wesley, 1995.

[4] BISCHOFBERGER, W. R., KOFLER, T., MÄTZLER, K.-U., SCHÄFFER, B., Computer Supported Cooperative Software Engineering with Beyond-Sniff; in: Proceedings Software Engineering Environments (edited by: Verrall, M.S.), IEEE CS Press, Los Alamitos, CA, pp. 135-143, 1995.

[5] BROOKS, F., No Silver Bullet. Essence and Accidents of Software Engineering; Computer, Vol. 20, No. 4, IEEE, April 1984.

[6] BURGER, C., Groupware: Kooperationsunterstützung für verteilte Anwendungen, dpunkt.verlag, 1997.

[7] DEWAN, P., RIEDL, J., Toward Computer Supported Concurrent Software Engineering; IEEE Software, IEEE CS Press, Los Alamitos, CA, 1993.

[8] DOURISH, P., Developing a Reflective Model of Collaborative Systems, in: SCM Transactions on Computer-Human Interaction (TOCHI), Vol. 2, No. 1, 1995.

[9] FLOYD, C., Software Engineering – und dann?, in: Informatik Spektrum, Band 17, 1994.

[10] FRIEDRICH, J., Defizite bei der software-ergonomischen Gestaltung computerunterstützter Gruppenarbeit, in: HARTMANN et al., Menschengerechte Groupware, Teubner, Stuttgart, 1994.

[11] GHEZZI, C., ARMENISE, P., BANDINELLI, S., MORZENTI, A., Software Process Representation Languages: Survey and Assessment; in: Proceedings of the 4th Conference on Software Engineering and Knowledge Engineering, IEEE CS Press, Los Alamitos, CA, pp. 455-462, 1992.

[12] HESSE, W., FRESE, M., Zur Arbeitssituation in der Softwareentwicklung, in: Informatik Forschung & Entwicklung, Band 9, Heft 3, 1994.

[13] HRUSCHKA, P., Herausforderung großer Projekte, Atlantic System Guild, Aachen, 1997.

[14] KRAUT, R. E., STREETER, L.A., Coordination in Software Development; in: Communications of the ACM, Vol. 38, No. 3, ACM Press, New York, NY, pp69 - 81, March 1995.

[15] LEHMANN, M.M., Process Models, Process Programs and Programming Support; in: Proceedings of the 9th International Conference on Software Engineering, ACM Press, New York, N.Y. , pp. 14-16, 1987.

[16] OSTERWEIL, L., Software Processes are Software too; in: Proceedings of the 9th International Conference on Software Engineering, ACM Press, New York, NY, pp. 2-13, 1987.

[17] PIEPENBURG, U., Ein Konzept von Kooperation und die technische Unterstützung kooperativer Prozesse in Bürobereichen, in: FRIEDRICH, J., et al., Computergestützte Gruppenarbeit, Teubner, Stuttgart, 1991.

[18] POMBERGER, G., BLASCHEK, G., Software Engineering, Hanser, 1996.