Cooperative Software Development: Concepts, Model and Tools

Josef Altmann, Gustav Pomberger

C. Doppler Laboratory for Software Engineering
Johannes Kepler University Linz
Altenberger Str. 69, A-4040 Linz, Austria
{altmann, pomberger}@swe.uni-linz.ac.at

Abstract

The development of large software systems demands intensive cooperation among multiple project team members with different responsibilities. The development process is often distributed across time and space and takes place within and between specialized workgroups. This necessitates finding appropriate answers to questions related to division of labor, to communication, and to coordination and cooperation in the planning, development and maintenance of software systems.

Development environments that explicitly support group work are an important prerequisite for the production of high-quality software systems. Most of the software development environments in use today support primarily technical aspects and have shortcomings in the area of organizational support. This paper describes a model for cooperative work processes in software projects and a corresponding development environment that provides balanced support for both organizational and technical aspects of software development.

The work toward the conception of the model and the implementation of the development environment have been completed, and the evaluation of the proposed approach has begun. Experience to date with using the developed environment confirms the assumption that the cooperative, cluster-oriented development improves both productivity and quality. The presented approach excels primarily in its easily understandable model, the intuitive usability of the tools and the comprehensible presentation of process- and product-related information.

1. Introduction

The prerequisites for the economical and professional production of high-quality software products extend beyond technical aspects and encompass in particular the process of distribution of development tasks and activities within a project team, the establishment of communication and coordination relationships, and the support of creative problem-solving processes.

Due to the spatial and temporal distribution of a large software project, collaboration between project team members is often asynchronous. Regular synchronization and the ability of developers to record momentary states and events are essential for efficient collaboration. Important aspects of supporting cooperative software development include the following:

• spontaneous collaboration
• direct and indirect communication
• building of group consciousness
knowledge about details and particulars of the project and its current state
the reuse of experience-based knowledge

Such aspects influence the economy of the development process and the quality of the target product.

A survey of current research approaches and the development environments used today reveals that group-supportive aspects are underrepresented or often fail to meet the requirements for efficient support of communication and coordination [1]. Although some tools do support group work in software projects (e.g., electronic mail, workflow management systems, group editors, synchronous debugging tools, etc.), these are generally not integral components of a development environment or they support only selected activities.

Process-centered software engineering (see, e.g., [2]) or approaches from computer-aided software engineering (see, e.g., [3]) strive to formally describe complex processes of software development in order to enable their (partial) automation. The prerequisite is a precise description of the software development process. However, the development process normally evades adequately precise a priori description due to continuous dynamic change; therefore the formalized process model approach seldom achieves success in practice. Furthermore, the application of strictly formal process models and ensuing automated control instruments fails to reap the potential of creative processes that can be decisive for product quality.

Beginning with a problem-oriented view of software projects that enables enumerating the significant characteristics of distributed, cooperative software development, this paper derives universal design principles for a work environment to support cooperative, distributed software projects. The identified design principles serve as the basis for the development of a model for cooperative work processes, which forms the starting point for the analysis, structuring, management and synchronization of divided-labor software development tasks. Next the paper provides an overview of a group-supportive work environment for cooperative software development; this enables the practical implementation of the developed model for cooperative work processes in software projects. The summary contains a short report on our experience with the use of the developed process model and the supportive development environment as well as a preview of planned research activities.

2. The role of cooperation in software development processes

Starting with a general view of the cooperation aspects of distributed teamwork, this section introduces the terms communication, coordination and cooperation. Next, the general characteristics of software development processes and finally the specific characteristics that distinguish distributed, cooperative software development processes are discussed with regard to the special needs of communication and cooperation.

2.1. Cooperation aspects of distributed teamwork

Cooperation processes such as software development in a team demand coordination processes in order to coordinate the cooperative activities of the individual team member with one another. Coordination processes in turn demand communication processes, such as the exchange of information across the various development activities. Based on [4], we use the terms communication, coordination and cooperation as follows:

Communication encompasses the process of transfer and exchange of information that takes place between communication partners. If the exchange of information serves to coordinate activities among team members, then this type of communication serves as the basis for the coordination of processes.
Coordination is based on suitable communication processes and encompasses all activities that are necessary for coordinating divided-labor tasks in the realm of a work process. While those involved in coordination processes pursue different goals, in the realm of cooperation multiple persons work together in a planned and coordinated way toward a common goal.

Cooperation is the manner of coordination that is necessary for agreeing on common goals and for the coordinated achievement of common work results among the participants.

Discussions of the concept of cooperation show that the organization, coordination and control of divided-labor tasks occurs on the basis of communication and coordination processes. Depending on the degree of freedom of the involved team members, we distinguish different forms of cooperation. We identify the following forms of cooperation on the basis of the influence and organizational possibilities of the participants:

- Team cooperation is characterized by the influence on the part of the other cooperation partner on the activities to be carried out. There is freedom in the establishment of internal organizational structures and process flows. The cooperative work occurs on the basis of agreed-upon goals and negotiated rules of cooperation. The cooperation partners are equals.

- Structured cooperation is enforced by formal mechanisms. Collaboration occurs primarily on the basis of objective and technical elements of the service generation process. Standardized workflows and formalized processes lead to external coordination and restriction of the alternatives for action. Cooperation routes are largely prescribed along with rigid spatial and temporal assignment of workplaces and work tasks. The cooperation partners occupy different hierarchical positions and are normally not equals.

The effects of structured cooperation on the software production process have been designated by Pasch [5] as software bureaucracy. A distinguishing characteristic of this bureaucracy is the formal embedding of the individual developer in a rigid organizational structure. Activities are coordinated and controlled by an external position (e.g., process modeller, workflow management system). The process of negotiating interests and resolving conflicts is largely suppressed.

Team cooperation supports unstructured, unplanable task completion and self-organization in a cooperative work context. Conflict serves as the starting point for a continuous process of coordination that can be guided and controlled by the participants. Team cooperation supports individual action and creativity and accommodates the current work context. In contrast to structured cooperation, computer-aided tools of the teamlike category are characterized by a supportive view. This view enables the participants to build mutual understanding of the cooperative work. The automation of a work process is not in the foreground.

The above discussions have shown different perspectives of cooperative work. The following subsections discuss first the important characteristics of software development processes and then the characteristics of distributed, cooperative software development processes.

2.2. Characteristics of software development processes

The software development process is significantly influenced by the underlying process model. A process model abstractly describes the general sequence of the development process along with its subtasks, their dependencies, and the targeted
results as well as the quality assurance measures that are used. The superordinate work steps are called phases. In the realm of software engineering, various phase models have been developed and compared. One of the main problems in the application of phase models is the distinction of the individual phases from one another and the consideration of the mutual effects between the phases. This paper does not describe the various phase models, but refers instead to the literature [3], [6], [7].

However, we do consider Meyer's cluster model [7] because on the one hand it refers to the object-oriented software technology in practice today and on the other hand by subdividing a project among various development teams it seeks to help avoid the main problems in the use of process models. As with most process models, Meyer's distinguishes among various project activities such as analysis, design, implementation and maintenance. In contrast to traditional phase models (e.g., waterfall model) sequential activities are not considered to be independent, isolated steps but as subsequent system extensions. Figure 1 depicts this development process according to Meyer, which reflects the essence of object-oriented development.

The software development process is not primarily characterized by phase-orientation and the resulting sequence of intermediate products, but by the development of components. The development team is responsible for all phases, and the strict phase distinction is relaxed; that is, the borders between the phases often disappear. This type of organization of the development process demands intensive communication and coordination within and between the development teams of the individual project clusters, because, to achieve flexibility and to exploit creative potential, neither the phase results within the cluster nor the cluster results are completely and formally specified in advance.

![Figure 1. Meyer's cluster model [7]](image-url)
completion of the decomposition of the overall project, for each cluster a mini-lifecycle is
defined. The subprojects can run simultaneously or staggered over time. Various project
goals are achieved at different deadlines. At the end of each subproject comes a
generalization phase in order to identify reusable software components and then make
them available across cluster and project boundaries. This results in a parallel or
overlapping development process that demands intensive communication and coordination
of the involved development teams.

Floyd and Züllighoven [8] distinguish between product- and process-related
development activities in software production.

- Product-related activities encompass requirements definition and system development.
  In the foreground is the software product with the corresponding development results,
  consisting of prototypes, programs and documents. The production process can be
  structured on the basis of predefined phases with predefined intermediate results.

- Process-oriented activities serve the purpose of coordination and cooperation in the
development process and encompass product management, quality assurance and
  project coordination. In a running project, reference lines are introduced for the purpose
  of assuring intermediate results. Reference lines define project states that the developer
  and user have agreed upon for synchronization of their cooperative work processes.

One problem is that software development is often viewed from only one of the
two perspectives. The product-oriented view focuses on the divided-labor software
product with its specific document types and system versions. By contrast, the
process-oriented view encompasses the divided-labor development process, which is
reflected in various documents. Thus an integration of the two views is appropriate
to master the problems in and the complexity of software projects.

Based on this basic understanding of the software development process, below we
outline the characteristics of software development projects. This outline is based on
literature studies as well as experience and observations with our own software projects.

- Rising complexity: The requirements on software products are continuously rising. The
growing complexity of software systems and the constant extension and adaptation to
new requirements require the cooperation of multiple persons. The size and complexity
of most software projects forces a distribution of the project tasks. Collaboration occurs
in specialized teams and among various subprojects and areas such as programming,
project management and quality assurance.

- Processes that cannot be formalized or automated: Software development differs from
the production of most other products in that the proportion of required creative
activities is significantly higher than the proportion of required reproductive activities.
In creative work processes the result is not defined in advance. The multiplicity of
software development tasks and the problems to be encountered require a high measure
of flexibility, creativity and continuous learning processes [9].

- High risk potential: Due to the usually incomplete or changing requirements and due to
the difficulties in the decomposition of the project tasks, the planning and execution of
a software development project is accompanied by uncertainties and risks [10]. The
number of collaborating subprojects or project groups whose development activities
have mutual influence cannot be determined at the start of the project.

- High communication requirements: In contrast to other production processes (e.g., for
hardware systems), due to the characteristics listed above, especially informal
information exchange among the project team members is of particular importance. In
contrast to formal communication, which is specified by a process model its defined
results, Bischofberger et al. [11] indicate the necessity and importance of informal communication during the project. The distributed shared decision making and the coordination in the team usually require spontaneous and flexible exchange of information between software developers.

- **High documentation requirements:** In addition to product documentation, the documentation process also requires process documentation in order to assure the comprehensibility of development decisions. The high degree of freedom and the low measure of available proven theoretical concepts does not always make design or implementation decisions readily comprehensible. Making use of knowledge about technical and administrative decisions, difficulties and their causes and corrections, conversations held between software developers, and modification and error statistics is an important prerequisite in order to support collaboration among the developers.

In an empirical study of 29 software projects in 19 firms, Hesse and Frese [12] determined that the effort for coordination processes, e.g., to exchange information, consumes ca. 40% of the overall development effort. The greater the dependency of development activities of various project team members, the more intensively they must cooperate with one another, inform one another, and keep each other up to date.

### 2.2. Characteristics of distributed, cooperative software development processes

The considerations of the general characteristics of the software development process in the previous section make it clear that software systems of a certain complexity cannot be realized in monolithic form, but must be decomposed into subsystem. The associated task-related spatial and temporal distribution of a cooperative software development process has the following characteristics:

- **Task-related distribution:** The rising complexity of software systems requires task-related decomposition of a project into subprojects as well as specialization within a development organization extending beyond project boundaries. In accordance with their capabilities, staff member and teams are assigned to specific subprocesses in software development. This demands a suitable coordination strategy.

- **Temporal distribution:** Depending on the scope of the project, subprojects are handled sequentially or in parallel. Various subprojects can be handled simultaneously. Various project goals are achieved at different times and so require synchronization or coordination of the work processes.

- **Spatial distribution:** Beyond the task-related and temporal distribution of development activities, there is also a distribution of project activities to different teams that, for organizational and/or economic reasons, can be spatially distributed. Spatial distribution requires providing a suitable communication infrastructure to cover the communication, coordination and cooperation requirements.

Based on the general concept of cooperation and the corresponding characteristics of distributed software development processes, we define the term cooperative software development as follows:

**Cooperative software development** encompasses covering the communication and coordination requirements within a software development process that are necessary for the planning, execution and coordination of all task-related, spatially and temporally distributed activities. Accordingly, cooperative software development encompasses all process- and product-related activities on the part of all participants whose common goal is the production of a software product.
Distributed, cooperative software development requires not only organizational support but also adequate tool support. In the area of computer-supported cooperative work, recent years have seen the development of various tools to support cooperative work. These systems generally fail to offer adequate support for the cooperative software development because they usually lack sufficient conceptual and technical integration of process and product views and insufficiently support the flexible, informal exchange of information.

3. Design principles for tool support

On the foundation of characteristics of distributed, cooperative software development processes introduced in Section 2, this section derives the design principles for the support of group work in software projects. The identified design principles form the basis for the development of a model for cooperative work processes in software development projects, and the model forms the basis for the implementation of a development environment to support cooperative software development processes. These design principles are the following:

- Transparency: The term transparency can be considered in the context of software engineering or in the application domain; here we limit the scope to the application domain. Applied to the support of cooperative software development, this means that the software developer is informed about the location of archives for artifacts, the distribution of development activities, and the competition situation with respect to other cooperating project team members. The project team members should have a comprehensible picture of limitations of resources and their spatial, temporal and task-related distribution. The coordination processes and the effects of access to artifacts should be clearly recognizable.

- Individual and group-oriented views of the current work process: Due to the task distribution, the project team members need an individual view of the current development process in order to achieve an overview of their responsibilities for certain activities. In addition, a group-oriented view is necessary for the cooperating software developer in order to be able to evaluate the activities in the overall context and to synchronize or coordinate the work processes in the team.

- Shared work space: The purpose of a shared work space is to provide and consistently manage process- and product-related artifacts. In addition, the team members must have an available cooperation environment in which to organize, structure and carry out their work. The organization of the work space includes the identification of the associated team members, the working materials and the tools that can be used for collaboration. The introduction of roles and their associated privileges and duties enables differentiation among team members of a work space and regulates access to artifacts.

- Promoting group consciousness: Building group consciousness requires making the structure of the development process clear to all software developers. This requires a notification service that automatically informs the project team members about completed and running activities and enables an overview of the activities of the cooperation partner in a software project. Dourish and Bellotti [13] call such consciousness about individual and group activities awareness.

- Negotiability of a conflict resolution: Conflict situations (e.g., competitive access to artifacts) should not be regulated completely by formal and automatable workflow descriptions. On the contrary, the cooperative support environment should be conceived such that control over the cooperative work process is left completely with the cooperating software developers and the responsible use of freedom of action is permitted [8]. The nature of the cooperation in a given development process can vary
according to priorities and conditions. Cooperation models and conventions describe and regulate collaboration within a software development team.

- **Availability of predefined workflows**: The workflow sequence of selected development activities in cooperative work processes should be made available in the form of schedules, directions and process patterns that describe the workflow of a task (e.g., code inspection). This is intended to support the cooperating persons in their mutual coordination and execution of work steps and in the determination of responsibilities. In the sense of Suchman [15], such coordination schedules should not be viewed as algorithmic, executable rules and regulations. Coordination schedules or predefined workflows should represent an outline and orientation for the divided-labor execution of a task – without a closed workflow in the formal sense and without the need for strictly following such a workflow.

- **Explicit and implicit exchange of information**: Explicit exchange of information among software developers occurs by sending messages, project reports, comments, etc.; the cooperation medium can be electronic mail or electronic bulletin boards. Implicit exchange of information needs to take place when changes occurred in the artifacts of a shared work space. Such changes should be propagated automatically by tools and made visible in the shared work environment. An event or signal mechanism notifies affected project team members of changes in shared artifacts; this notification could be in the form of a message or via appropriate representation in the user interface of the affected team members’ development environment.

- **Transparency of the process history**: A process history evolves during the development process; this history should be appropriately documented and made available for later access by the project team members. This documentation should contain information about the activities carried out, problems encountered, dependencies on other activities, alternative solutions, knowledge gained, etc. This improves overall understanding and assures efficient problem solutions through reuse of the project team members’ knowledge from experience.

- **Status and context information**: The status information about a project that team members require can be either organization-related or project-specific in nature. Organization-related information includes the composition of the work groups and their responsibilities. Project-specific information concerns the status of activities, causes for modifications and instructions for the programming of extensions. A cooperative support environment should provide such status and context information to facilitate the project team members’ orientation in the current work context.

From these design principles, the following section derives a model for cooperative software development processes. The proposed model forms the basis for the implementation of a corresponding cooperation environment.

### 4. A model for cooperative software development processes

The goal of the model is to describe development activities along with the associated project team members and the relationships among the workflows. Figure 2 depicts the components that are most important for understanding the overall model (in OMT notation).
Figure 2: Model of cooperative software development processes

The model comprises two logically connected areas:

- **Process view**: The components for mapping the process view, which belong to the core of the model, form the basis for coordination and cooperation during the software development process. The process-related model describes the tasks of the development activities and their relationships. This also includes assigning the process participants to the individual tasks and ensuring that the development process is recorded.

- **Product view**: The product view includes the results of a software project. The development results consist of documents, executable programs, prototypes and software libraries.

The following subsections introduce the components of the model and explain the important terms in this context.

4.1. Process view

The core of the model is the process model. The process-related structuring of a software development project occurs via decomposition of an overall task into multiple smaller subprojects and subtasks. A subtask is represented in the model by a work package and consists of one or more work steps. A work context describes a subproject and manages a group of logically associated work packages. Multiple work contexts together form a shared work space that combines all development activities of a certain subproject. The shared work space serves the team members as an entry point into a software development project and forms the basis for the coordination and cooperation of the development activities. Depending on their roles, the individual team members can access the process- and product-related information and inform themselves about the status of the development process. The activities that are carried out with individual artifacts are visible to certain participants, depending on the configuration of a work space, and are recorded in a process history.

After the basic concept for the decomposition of a software development project has been defined, selected components of the model are discussed to enhance understanding.
• **Process model:** A process model in the sense of the proposed model defines the basic structure of a development process, which consists of prefabricated work contexts, work packages and work steps. As a rule the process model is not re-invented for each new project. Generic process models for various types of software projects are stored in a process model catalog and are available as starting points for launching new development projects.

• **Work step:** A work step represents an activity within a work package. A work step is represented by its designation, a description, the process status, and a link to the next work step. The description contains work instructions for carrying out the work step. Template documents can be provided as input documents. A work step can be refined via other work steps. This creates a hierarchical structure of work steps that represents a work schedule for processing a work package.

• **Work schedule:** A work schedule consists of work steps. In this model the work schedule serves as instructions for activities or as a hierarchical guide to support the processing of a work package. Work schedules are not process regulations that completely control the activities of the project team members. Depending on the respective project conventions, which are established at the start of a development project, deviations from the prescribed work schedule can be justified by project team members and documented in the respective work steps. Proven work schedules (best practice) for various development activities (e.g., error correction, integration test) are collected in a catalog and made available for the creation of other work packages.

• **Work package:** A work package describes a goal-oriented processing order that forms a logical unit and is executed by a project team member. The creator of a work package can be the one to handle the work package or can delegate it to a member of the project team. A project team member can handle the assigned work package, reject it, or after coordination with the creator delegate it to another team member. The coordination of the work process as well as the assignment of access privileges for parts of a work package are the responsibility of the creator. An evaluator is responsible for checking and receiving the results of a work package. The creator bears the responsibility for the achieved results and coordinates the handling of corrective suggestions between the evaluator and the processing team member. A work package description contains specifications concerning the creator, the processor, the evaluator, the role(s) that the processor must assume in order to be allowed to handle the work package, the types of tasks, the priority, the completion status, the date of creation, the planned and actual date of completion, the modification dates and the evaluation date. The documentation of work progress, the achieved results, and the results of the evaluation process are appended electronically to the work package description by the respective project team members. Every work package contains a process history that consists of a list of executed work events. The work package is the most important medium to support collaboration among the project team members. The exchange of work packages constitutes the divided-labor software production process. Additionally, annotations and package-related messages support informal cooperation in a development team and the documentation of development decisions.

• **Work context:** In a work context, logically related work packages are managed; the work context serves as a hierarchical structuring medium for a work space. A work context in turn can be refined by means of additional work contexts. The creator of a work context coordinates the sequence and execution of the work packages and bears the responsibility for the results of the work context. The evaluator of a work context checks the overall result of a work context, which comprises the component results of
the individual work packages. The creator of a work context delegates the corrections suggested by the evaluators to those responsible for the respective work packages and monitors the timely and correct execution of the revisions. Work contexts divide a shared work space into various areas of interest. This promotes orientation in cooperative development processes, since coordination and cooperation are no longer restricted to individual work packages, but occur in a larger context, the work context.

- **Work space**: A work space encompasses multiple development activities that are necessary to achieve the common goal of one or more project groups. A work space is structured hierarchically into various work contexts and work packages. The description of a work space contains all information necessary for planning and executing a project. The most important artifacts are the project order and the description of the project organization. The structure of a work space, consisting of work contexts and work packages, is continuously complemented and updated by the project team members. In certain intervals the project leader checks the work space structure and adapts it to the current process. A prescriptive description of the development process is not supported by the proposed work space, since this would not correspond to the low a priori specification and high complexity of software development processes.

4.2. Product view

The product view relates to the software product to be developed. The software product comprises programs, libraries of reusable software components, prototypes, and development documents. The product view is represented abstractly by an artifact that can occur in multiple instances with various forms.

**Artifact**: An artifact represents a work object in the software development process. Both the executable program system itself and all documents of the development process are artifacts.

This section has presented the base model proposed for modeling (i.e., structuring and describing) a software development process. In the foreground are the concretization of the responsibilities of the individual project team members, the mutual coordination of the work processes, and the description of work scenarios.

5. A work environment for cooperative software development

This section introduces a work environment for cooperative software development that enables practical application of the model for cooperative work processes in software projects described in Section 4. The work environment supports spatially and temporally distributed project team members in the context of typical software development activities.

The work environment Cooperation Assistant was developed as an experimental platform at the C. Doppler Laboratory for Software Engineering at Johannes Kepler University of Linz [15]. The concepts implemented in Cooperation Assistant to support distributed, cooperative software development strive to harmonically unify the product- and process-related views. The environment is based on the approach of shared work spaces. A task to be cooperatively handled is represented in the work space as a work package.

Introducing all the tools of this extensive work environment and discussing implementation details would exceed the scope of this paper. Hence we present the Cooperation Assistant only as an overview on the basis of an application scenario. For reasons of compatibility with other integrated software development tools and due to the wide usage of the language, C++ was chosen as the implementation language. As implementation platform we chose the framework ObjectWire [16], which was developed...
at the C. Doppler Laboratory and particularly supports the implementation of distributed, object-oriented software architectures.

Figure 3 shows an overview of the selected tools of the cooperation environment from the viewpoint of a project team member. The Cooperation Assistant provides tools for structuring, monitoring and processing cooperative work processes. Due to space considerations, we restrict our description to the tools Agenda, Work Context Manager and Work Package Editor.

Figure 3 shows the project structure of the development project ObjectWire (in progress), which is depicted in the Agenda. The presented structure of the software development project gives project team member Josef an overview of the logical associations of the individual development activities of the project group. The overall project in this application scenario consists of the subprojects Monitoring, Configuration, and Cooperative Software Development, as established by the project leader at the start of the project.
Monitoring the sequence and work progress of the development activities of a subproject occurs in the Work Context Manager. Figure 3 shows a Work Context Manager that represents all current and completed work packages of the work context Configuration. On a state transition of a work package, the event service of the cooperative work environment creates an event object. For example, a state transition in the creation, sending, rejection or acceptance of a work package. The reporting service informs the project team members about the events that occur. The events are depicted graphically in the Work Context Manager alongside the affected work packages; e.g., the symbol means that the work package has been modified.

The Work Package Editor supports collective processing of development tasks. The description of a work package reflects both the description and the processing of a work package. The Work Package Editor provides tools for processing a work package and to support communication and coordination among the staff members of work spaces. The connection of the process view with the product view occurs via integration of existing software development tools. In the application example, TakeFive's programming environment SNiFF [17] and a design tool developed at the C. Doppler Laboratory were integrated into the Cooperation Assistant. The tools available for the product view might vary from project to project.

Work package descriptions, work schedules, annotations and comments are put in relation to one another and to documents in the form of links. The links inserted in the description of the work package complement the task specification with the software development documents that are necessary for processing the work package; activating a link invokes the external application. In the application scenario there are links to locations in the implementation and in the design that are in relation with the task description of the work package. From there the software developer can begin processing the task in the respective software development tool. The results of this processing are documented in the Work Package Editor by the package processor. On completion of processing, the work package is marked as completed and automatically forwarded to the respective evaluator, who was specified with the Work Package Editor.

The above application scenario shows that the model for cooperative, distributed software development processes from Section 4 has been implemented in the Cooperation Assistant. The Cooperation Assistant supports collaboration through the exchange of shared work objects and covers the complete software development process. General functionality for process-oriented management of cooperative workflows was linked to the artifacts with a product-oriented view. Likewise, external software development tools can be integrated into the work environment with the help of tool adapters. The authors know of no other available software development environment that equally supports both the process view and the product view; also provides flexible communication, coordination and cooperation mechanisms; and additionally permits the integration of external tools.

6. Summary and perspectives

The proposed model for cooperative software development and the development environment Cooperation Assistant based on this model serve as a contribution toward correcting the shortcomings of software development environments; in addition, they provide an experimental environment that permits empirically-based support or rejection of the thesis that the cooperative, cluster-oriented development approach is superior to the rigid phase-oriented development approach with respect to exploitation of productivity and improvement in quality potential, especially for spatially and temporally distributed project organization.
The work toward the conception of the model and the implementation of the development environment have been completed, and the evaluation of the proposed approach has begun. Experience to date with using the Cooperation Assistant confirms the assumption that the cooperative, cluster-oriented development improves both productivity and quality. The presented approach excels primarily in its easily understandable model, the intuitive usability of the tools and the comprehensible presentation of process- and product-related information.

The evaluation research activities have not been completed yet, and the presently available analysis data do not suffice for empirical evaluation and conclusion with respect to support or rejection of the above thesis. Evaluation will continue to take some time, since only the results of a correspondingly large number of extensive, complex software projects can provide the basis for such evaluation.

Open issues that are the subject of continuing research and development work include the integration of network techniques for the control and monitoring of tasks and deadlines, the integration (acquisition and use) of metrics for process optimization, and the conception and availability of tools for user-controlled analysis, filtering and sorting of product- and process-related information.

7. References