Communication, Collaboration and Cooperation in Software Development - How Should We Support Group Work in Software Development?

Motoshi Saeki
Dept. of Computer Science
Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo 152, Japan

Abstract

Software development is essentially cooperative work which is collaboratively performed by various roles of persons and tools. Communication among the members of a development team, e.g. conversation, and among tools is one of the most important characteristics for these collaborative work. To make our software development environment more effective and comfortable, we should observe what communication, collaboration and cooperation are actually made in development processes and what styles are suitable to us. In the paper, several case studies and analytic results in software development are surveyed, and I discuss what kinds of tools are required to support seamlessly group work in software development.

1 Introduction

Software activities are essentially cooperative and performed a collaboration of a team. In such cooperative situation, different roles of workers such as customers, users, analysts, designers, managers, and so on, actually participate in the activities and communicate with each other.

Conventional CASE tools for supporting software development[21] are for a single developer. Thus it seems to be difficult to apply effectively the existing CASE tools to the activities which are performed by a team. The members cooperate, collaborate, coordinate, and communicate with each other to develop an artifact.

Providing the support tools for cooperative work is the main problem in CSCW (Computer Supported Cooperative Work) and Software Engineering communities. They have studied this topic independently and produced research results. CSCW community focuses on the tools only for general purpose, e.g. electronic meetings, decision making support, collaborative editing, and so on. However they are for general cooperative work and the community does not consider special applications such as software development yet. As Curtis pointed out in[9], specialization to the application of software development allows us to have more effective supporting tool.

Recent research in Software Engineering community, especially Software Process community has produced process-centered software develop environments (SDEs), e.g. Merlin[29] and Marvel[1]. Supporting cooperative work in them is mainly concurrency control of multiple access to a product to maintain consistency and to avoid interference. That is to say, the CSCW community and the Software Engineering community don't have much intersection both on researchers and on research directions. In this paper, I report the current status of research results of CSCW and discuss their applicability to software engineering field.

The organization of the paper is as follows. In the next section, I report the technique to observe and analyze practical cooperative activities and then illustrate empirical case studies done by software engineering community. Observing and analyzing "actual" cooperative activities are very important to clarify their characteristics. Cooperation style or characteristics depends on each phase of software development, e.g. requirements elicitation, specification development (analysis & design), coding, testing phases and so on. Thus to develop effective supporting tools in a phase, you should observe the actual activities in the phase and extract its characteristics such as obstacles to cooperation. The past failure of applying the existing groupware resulted from less empirical studies on the actual and practical software development processes. Section 3 surveys groupware tools in CSCW community. The overview of SDE supporting tools developed by software engineering community is reported in the next section. In addition, I introduce the examples of applying the research results of CSCW to software development support, including our current study. In the final section, I discuss the current issues and the future directions in the two communities.

2 Observing and Analyzing Activities

2.1 Techniques for Observation and Analysis

The aim of CSCW is to clarify cooperative activities of members in the group that has a goal to be achieved and how to support them with computer technology, while *groupware* denotes a computer-based system supporting cooperative activities. Researchers from various fields, e.g. psychology, sociology, linguistics, cognitive science, ethnography and, needless to say, computer science, are involved in CSCW. Although the final goal of CSCW research is computer support for human cooperative activities, many of them focus on human behavior under group environments, e.g. how to communicate and negotiate with other members, awareness among the members and so on. As I mentioned before, clarifying human cooperative activities is very important to achieve an effective support. What kind of work a group perform varies its cooperative activities. For example, the activities in software development are different from in program committee meetings of an academic conference such as APSEC'95. Thus we need empirical and analytic studies of actual cooperative activities in our specific domain.

In CSCW research, with the aid of social scientists and cultural ones, several methodologies for empirical and analytic studies have been applied to the actual human activities. First of all, we should observe them and clarify how to do, i.e. what characteristics in them we focus on. From engineering sense, we preferably try to find the obstacles that prevent cooperative activities and reduce productivities. Almost of all existing studies in CSCW community observed communication among the group members, especially conversation made by them. In a face-to-face meeting, which is a typical example of the forms of cooperative activities, the participants spend almost all of meeting time in verbal conversation. As this fact suggests, verbal conversation can be considered as a key factor to clarify cooperative activities.

Conversational analysis is one of the techniques to analyze conversation records. This technique captures conversation as a sequences of utterances, and focuses on the syntactic characteristics of the sequence to explore its structure rather than on the contents of the utterances. For example, to extract turn taking, i.e. changes of topics being discussed, the technique suggests that emphasis should be on time interval of silence between a temporal adjacent pair of utterances.

The representative techniques based on the contents of utterances are *content analysis* and *speech act theory*[36]. In these techniques, the analysts classify utterances into pre-defined semantical categories based on the contents of utterances. Understanding completely the utterances to be analyzed is needed and it leads to huge time-consumption for analysis. Furthermore an analytic result depends on the workers who perform actually the analysis although they analyze the same utterance record. The reason is that the human utterances contain semantical ambiguity. That is to say, this type of analysis has the possibility of getting unstable results.

2.2 Empirical Case Studies in Software Development

There are several empirical studies on observing and analyzing a part of practical software development processes and they concentrated on requirements elicitation or specification development phase in conventional waterfall process model.

Walz, Elam and Curtis analyzed series of the meetings to design an object management servers[9, 38]. This design phase contained 19 meetings in and two of these meetings were seminars by outside experts to acquire technical knowledge. It ceased in four months and all the meetings were videotaped. A project manger, eight designers and a customer group were involved in the design meetings. However all of them did not always participate in the meetings. By analyzing the utterances in the meetings, the participants spent much time in developing the common model of the object server that could be shared by them.

Olson et al. studied software design meetings from four development projects[28]. Each project contained several meetings – from two to five Each meeting spent time from an hour to two hours and from three to seven persons were involved in. They videotaped these meetings and coded, i.e. classified the utterances into some categories based on their contents. Their findings are that 40% of the meeting time was spent in direct discussion of design, 30 % in progress evaluation and 20% in team coordination.

The term "participatory design" recently often reach our ears and it means that users (and other stakeholders such as customers) work together with designers in software design processes. In the development using Prototype, the users and customers are always involved in the phase of assessing the produced prototype. Some empirical studies explore the benefits and shortfalls of user and customer participation in requirements elicitation and specification development phases[25]. Requirements elicitation and specification were originally the tasks that users or customers did not perform but developer did. The aim of the studies is to clarify what impacts the cooperation of the users with the developers have and the emphasis is on role of the participants in the tasks. The useful findings will be obtained increasingly as we have more experiences in practical participatory design processes.

These studies mentioned above focused on the characteristics of human behavior, not on the artifacts produced through cooperative work. Kaiya et al. investigated not only characteristics of utterances in the requirements elicitation meetings but also relationship among the utterances and the artifacts, i.e requirements specification[18]. They had an assumption that the utterances denoting conclusions should appear in some part of the artifact and they tried to relate the videotaped utterances to a part of the artifact. They found that about 30% of the utterances denoting conclusions lacked for the artifacts. Furthermore they analyzed typical patterns of utterance occurrences which often caused the lack of the conclusions in the artifacts.

As computer networks spread wider, E-mails as a communication tool become popular and are used in software development. The advantages and deficiencies in using E-mails in software design phase is explored from empirical studies[35]. The software designers in this experiment found the deficiencies in the following :

1. Text in the mails, who was composed by the others, might be not comprehensive to the readers.
2. Conclusion to which the members came might be ambiguous in the electronic discussion, or it might be unclear where the conclusion was in the set of the mails.
3. All of the members did not discuss a topic very much but the discussions might be one-to-one.
4. It is often delayed to read the messages. Delayed readers might be left behind or out of the discussions.

Furthermore it reported that the contents of the E-mails that the designers passed were classified as shown in Table 1.

The term "work structure" means the division of work in a group and Bendifallah and Scacchi investigated the work structures in software specification development[2]. They found four types of work structures; negotiative, integrative, delegative, and replicative work structures and typical patterns of the shift of work structures in actual specification development processes. In the case of negotiative one, a group concentrates on negotiation such as making a plan and a schedule and it produces products with the consensus of the group members. In the integrated work structure, the final product is an integration of subproducts that the members produced. In the delegative work structure, an individual of a group is delegated to produce a final product. Replicative work structure includes redundant and duplicated activities. More than one member perform the same task in the structure.

As I introduced above, empirical studies in software development field just started and I hope that this type of researches increase and we have more findings in actual cooperative activities. The techniques that these researches adopted is 1) recording human activities with videotapes, 2) making transcriptions from the utterance records, 3) classifying the transcriptions into some categories based on their contents, i.e. coding transcriptions, and 4) extracting their characteristics. One of the difficulties is the complexity of analyzing videotaped records of human activities. Although we ignore gestures and movement and only focus on utterances, we should listen to large volume of utterances and make transcriptions. And it may be difficult to keep stability of coding. To assess and increase the objectivity of coding, we can adopt more than analysts who code the same transcription and integrate their results by using some principle such as majority. The problem of complexity is serious because practical software development processes go in a long term. We should reconsider the granularity of the activities to be analyzed. However the significance of observing and analying cooperative activities in practical software processes will never be reduced.

3 What Does Groupware Support?

The next point is the question what is groupware, which part groupware can support. In the section, we look back the evolution process of human cooperative activities beginning with simple communication between human. Next, we will consider which part modern groupware supports

3.1 Evolution of Groupware

Human cooperative activities begin with communication among the partners. Communication is made by verbal conversation, letters, telephone, fax, E-mails through computer network and so on. Recently software for sending video and audio such as NV and VIC on computers through computer network has been developed. In this level, we just send and receive various kinds of information such as text, voice and figures to our partners. Note that we have two types of communication — synchronous communication and asynchronous one. In synchronous communication such as verbal conversation and telephone, the message that you sent can be immediately read by the receiver, while the messages you sent are stored in mail boxes and the receivers may be delayed to read it in asynchronous communication such as letter and E-mails. Furthermore the order of reading messages is not always the same as the order of sending out them.

To perform tasks communicating with the partners, a memory for storing intermediate artifacts is needed. The second stage in the evolution is adding a private workspace for each participant to communication facility. The participant accesses and updates his or her private workspace which is invisible to the others. The participant decides what communication should be made by investigating his or her workspace. As more complex task group members should perform or more larger artifact they should produce, they need

Table 1: Contents of E-mail Communications

Content	%
Discussion about Modification of Decided Matters	25
Comments & Response	23
Discussion about Undecided Matters	15
Notification of Progression, Schedule Negotiation	11
Confirmation of Decided Matters	6
Terminology Definition & Confirmation	6
Others	14

Total number of Mails : 71 Period : 14 days

the workspace where they perform the task cooperatively. It also stores intermediate artifacts leading to a final artifact. This workspace is shared by the members and all of them can access and update it. This form can be considered as modern groupware. That is to say, communication facility, private workspaces and a shared workspace constitute groupware. Figure 1 illustrates this evolution, i.e. from communication support to collaboration and cooperation support.

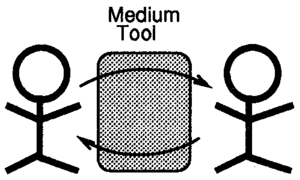
3.2 Architectural Structure of Groupware

Architectural structure of groupware is shown in Figure 2. It adopts layered architecture based on the evolution process mentioned in the previous subsection.

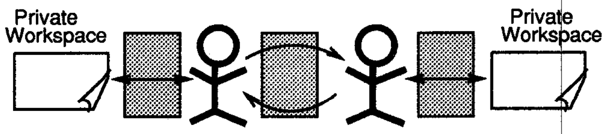
1. Communication : In this layer, communication tools and media are supported. Suppose that we use E-mail system on Internet. Network hardware such as Ethernet and ISDN, and software for communication protocol such as TCP/IP support our communication by E-mail. What kind of information you send, i.e. message structure depends on the layer. In the case that you send video images, this layer supports the coding techniques for data compression such as MPEG2 and H.261 etc. Synchronism of communication, i.e. synchronous communication or asynchronous, also depends on the communication tools and media you selected.

2. Private Workspace Management : The private workspace is used for storing messages from the partners and for composing messages. In the case of E-mail system, mail holders are a private workspace. It reflects the internal state of the worker and the worker can decide what action he or she should do from the state of his or her private workspace.

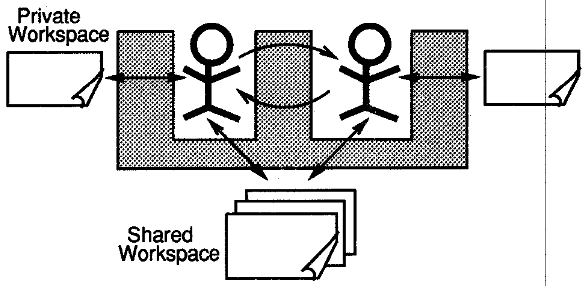
3. Shared Workspace Management : While a private workspace is invisible to others, a shared workspace can be accessed by all of the group members. In this situation, concurrency control



(a) Communication



(b) Private Workspace



(c) Shared and Private Workspaces

Figure 1: Conceptual Evolution of Groupware

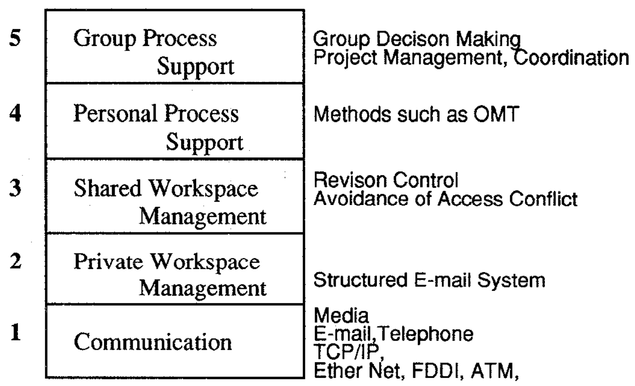


Figure 2: Architectural Structure of Groupware

of multiple access to a workspace is very important to maintain consistency and to avoid interference. Conventional locking techniques in database area such as two phase locking and check-in & check-out may not be suitable for management of a shared workspace.

4. Personal Process Support : We have many methods for software development, e.g. Structured Analysis/Design[10, 40], Object-Oriented Analysis/Design[32, 7, 37, 4] and Jackson Systems Development[16]. These methods should be supported in the layer.

5. Group Process Support : Before starting software development, a project leader makes a project plan. The plan includes development schedule, resource allocation and assignment of workers. After starting the development, the leader monitors the progress and may change the plan when the deviation is encountered. This type of coordination is called project management and should be supported in the layer.

3.3 Groupware Developed in CSCW

Communication :

VAT (Visual Audio Tool), NV (Net Video) and WB (White Board) are used for sending presentations in multicast way through MBone on Internet. Furthermore we have other tools to sending video and voice for desktop conferences.

Private Workspace Management :

Object Lens (currently its enhanced version Oval is developed) is a structured E-mail system and we can specify what action the system performs when an E-mail is received by a rule in template form[19]. For example, you can specify the rules so that the received E-mails are automatically stored in the corresponding mail folders according to the values of their "From" and/or "subject" slots. That is to say, it can support automatic classification and filing of the received E-mails.

Coordinator[39] is an E-mail system based on speech act theory and for making a commitment between two persons. E-mails sent in this system have type related to speech acts such as requests, promises, decline, counter-offer and so on. A state transition machine specifies which type of E-mails the receiver should reply when he or she receives a type of E-mails, i.e. legal conversation moves (e.g. after a requests is issued, a promise, a counter-offer or a decline can be made).

Shared Workspace Management :

Co-authoring systems or group editors such as Grove[13] and Quilt[5] focus on the shared workspace management, i.e. concurrency control of multiple access to maintain consistency. The simple solution may be locking to achieve mutual exclusion. The part which a person is editing should be locked so that others cannot access it. However, in fact it is not a simple solution. How large grains should we lock — a paragraph, a sentence, a word, a character or a key stroke level? When should we lock — moving a cursor or updating a character? Grove is for simultaneously editing an outline by a group and has no specific locking mechanism. The members can view any part of the text and update it at any time. It adopts cloud-burst model to inform the others which part a member is editing. When a member inputs the text, it appears in bright blue on the editor screen. It gradually changes black as the time passes by. While a member modifies the text, it is covered with a cloud on the screens of the other members. They know that someone are modifying the text by noticing the cloud. Of course, the member that initiates the modification can view the text, i.e. the cloud does not appear on his or her screen. The emphasis of Grove is on *awareness* of what the others do. Members themselves decide what they should do by seeing what the others is doing. This mechanism is very simple and it avoids adopting complex locking mechanisms. Furthermore this kind of awareness plays an important role on good cooperation among persons from the social viewpoint.

The next point is the structure of shared workspace. What data does the shared workspace holds? What structure should we use to hold the data? Some researches address that group activities are recorded to decide the next activities of the group. QOC (Questions Options and Criteria)[22], gIBIS[8] and SYBIL[20] specify the structure of discussion activities for design tasks. For example, the discussion activities are classified into three categories – Issue, Position and Argument in gIBIS model. By analyzing the records that are stored following the model, we can extract decision rationale and find the issues that we have not discussed yet. Structuring the activity records is use-

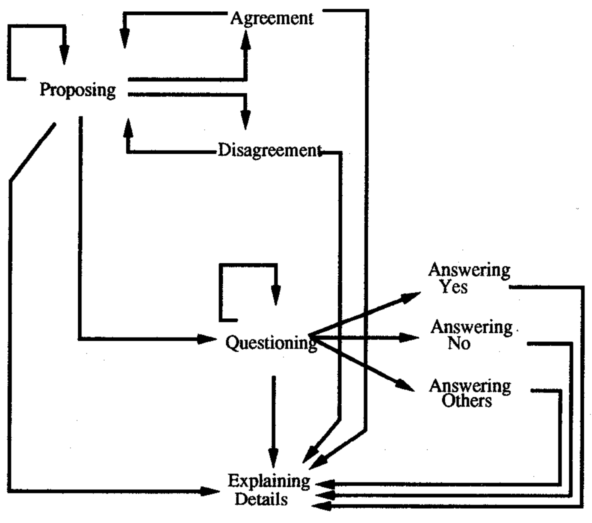


Figure 3: Typical Pattern of Speech Act Transition

ful for decision support. However, the members are forced to discuss anything following the model and it may restrict the members' activities. Thus we should construct the model natural to human behavior.

4 Applying Groupware to SE

This section introduces groupware or supporting tools developed by Software Engineering community. These tools support specific phrases in software development — requirements elicitation, specification development, coding, review, test and maintenance.

4.1 Requirements Elicitation

As I mentioned in the previous section, it is important to hold the discussion records in a structural way. The point is what structure we adopt as a database schema. Potts et al. proposed the requirements elicitation could be be considered as a inquiry-answer cycle between customers and developers[30]. They formulated a model expressing this cycle based on speech act theory and developed a tool for storing and accessing discussion records by using hyper text.

It is a problem whether such a model fits to human activities. In our study[34], we focused on the face-to-face meetings where a customer explains his or her requirements to the developers and construct a speech act transition model as shown in Figure 3. We investigated actual meetings and we could see that 84 % of the utterance transitions follow the typical patterns of Figure 3. This result leads to the feasibility of constructing speech act based model to hold the meeting activities in structural way. And this record is useful to develop requirements specification and to manage its modification[14]. Requirement elicitation may have several phases — the phase where the customers explain their requirements to the developers, the phase where the developers (the analysts) present their solutions to the customers, the phase where the participants discuss the solutions to come to agreement, the phase for negotiating development schedule, and so on. Thus the models should depend on the phases and we should develop the models in the other phases. How we should extract speech acts from the utterances is a problem. To label the utterances with speech acts, we focused on the keywords that characterize speech acts.

Next, how we use the records of the activities? The resolution of requirements conflicts among the customers' side and developers' is one of the important applications. Boehm et al. considered requirements elicitation as negotiation and renegotiation processes among the stakeholders such as users, customers and developers, called Win-Win Spiral Processes[3]. In the processes, each stakeholder captures his or her desired objective called win conditions at first. Next the stakeholders detect the conflicts between the win conditions and their specifications. And then they try to find the agreement conditions which satisfy the stakeholders' win conditions. The process to resolve the conflicts and to find the agreements is formalized with a state transition diagram and the supporting tool WinWin-1 has been developed. Robinson also proposed the technique for support negotiation processes of developers and customers[31]. Similarly to the Boehm's work, requirements from the customers and the developers are recorded as a dependency graph, which expresses the positive dependency relationships and negative ones (the relationships expressing conflicts or mutually exclusions) among the requirements. This graph is useful to reason the impacts to the other requirements when a group accepts or rejects a requirement.

4.2 Specification Development

We have many methods to develop specifications. However all of them are not for a group but for an individual worker. We need construct methods and the supporting tools suitable for group work or for the development organizations. Method engineering provides the framework to compose a method from the method fragments pre-stored in a database called method base[15, 33].

In specification development by a group, each worker may develop a part of the specification by the methods different from the others. For example, a worker uses a data flow diagram while the other one develops a state transition diagram. In this case, how to integrate the specification parts that the different workers developed into one and how to check inconsistency occurring in the integration are a problem to be solved[27].

4.3 Coding

In coding phase, the groupware that we often use is group editors. Unsimilar to the other work, in software development, there are workers responsible for the decomposed sub tasks. Only the responsible worker can develop and modify his or her software modules. Thus the concurrency control of accessing a source file can be achieved by the management of responsibility and the tools that do not show the modules to irresponsible workers[17].

4.4 Review and Inspection

Review and code inspection are usually performed in face-to-face meetings. Some researches on review and inspection by teleconference and by desktop conference were reported[6, 11, 24]. They suggested that teleconference and desktop conference are as effective as face-to-face meetings in review and inspection phase. However some devices for floor control mechanism such as FIFO might be needed to reduce network load and to have efficient meetings.

Almost all of the tools for review and code inspection have the functions that the others can annotate to a code on the screens of their terminals. In small teams, these tools are effective. However as the participants increase, the developer responsible for the reviewed or inspected codes would bother with the management of the annotations which were made from the different persons but had the same meanings.

4.5 Debug and Test

Individual workers who have responsibility the tested modules are involved at their sites in their tasks, i.e. debug and unit test of the modules. To discuss the detected bugs or the test result, they may communicate with the others, e.g. the reviewers who had reviewed and commented to their modules. To enhance communication, the communication mechanism should be integrated to the debuggers and testing tools. For example, MShell of Flesco are used for multi users, and they can enter the commands to the tool interleavingly and share the responses to the commands entered by them[11]. It allows all of the participants to see the bugs or the test results that the tools output.

After approving the results of unit tests, the modules are sent to and collected at the site where integration tests will be performed. As all of the persons responsible for modules can participate in the work, supporting tools for synchronous communication such as teleconference tools or desktop conference tools are preferably integrated to the tools for integration test.

4.6 Maintenance

Modifying software modules causes the problem of multiple access and updating, and it leads to inconsistency of the modules. In [23], revisioning modules is notified to the other workers by changing a color of the module on the screen of the tool which displays the structure of versions. Awareness among workers, i.e. notifying to the others what I do, is a simple but very powerful technique to avoid the multiple access in software development environments. However it is just useful for small groups.

There is another approach for supporting revision control. The developer that would like to change the module should propose it to the related persons at first. The change cannot be made until the proposal is approved by them. This mechanism is called "Lazy" consistency[26] and the tool supports the communication for proposal-and-approval protocol.

5 Summary and Future Directions

Table 2 summarizes the key technologies which are and will be very important for software development. They come from both groupware technologies and software engineering ones.

As I mentioned before, characteristics of group work in software development vary on the development phases. We should clarify the characteristics through empirical studies, establish the key technologies mentioned in Table 2, and develop the tools suitable for each phase. It is infeasible to support a single tool thoughout software development processes. That is to say, like Flesco[11], our tool may be an integration of the individual tools that support a phase, i.e. *integrated groupware.* To integrate the tools, it should have open architecture. A tool that generates groupware tools, so called *meta groupware*, may be needed. The technique to compose a suitable groupware from groupware tool fragments stored in a database, *groupware tool base* also becomes important. To achieve this goal, the study of the conceptual model or meta model of groupware such as [12] is one of our directions.

Acknowledgements

The author would like to thank Prof. Koichiro Ochimizu of JAIST, Prof. Keijiro Araki of NAIST and the members of the research group BUN3 for their invaluable discussions and suggestions to my work. The empirical studies described in the section were supported by Dr. H. Kaiya, N. Miura, W. Kuo and K. Matsumura and my thanks also go to them.

References

[1] N.S. Barghouti. Supporting Cooperation in the MARVEL Process-Centered SDE. In *SIGSOFT'92 : Proc. of the Fifth ACM SIGSOFT Symposium on Software Development Environments*, pages 21–31, 1992.

[2] S. Bendifallah and W. Scacchi. Work Structures and Shifts : An Empirical Analysis of Software Specification Teamwork. In *Proc. of 11th ICSE*, pages 260–270, 1989.

Table 2: Groupware Support in Software Development

Phase	Key Technologies
Requirements Elicitation	Group Decision Support, Conflict Resolution, Recording Activities based on Speech Act Theory
Specification Development	Method Engineering (Method for Group), Consistency Checking
Coding	Group Editing, Awareness
Review & Inspection	Teleconferencing, Desktop Conferencing
Debug & Test	Integration of Communication Tools to Debuggers and Testing Tools
Maintenance	Revision Control with Conflict Resolution, Awareness
Project Management	Group Decision Support, Awareness

[3] B. Boehm, P. Bose, E. Horowitz, and M. Lee. Software requirements Negotiation and Renegotiation Aids : A Theory-W Based Approach. In *Proc. of 17th ICSE*, pages 243 – 253, 1995.

[4] G. Booch. Object-Oriented Development. *IEEE Trans. on Soft. Eng.*, 12(2):211–221, 1986.

[5] U. Borghoff and G. Teege. Application of Collaborative Editing to Software-Engineering Projects. In *ACM SIGSOFT, Software Engineering Notes*, number 3, pages A56–A64, 1993.

[6] L. Brothers, V. Sembugamoorthy, and M. Muller. ICICLE : Groupware for Code Inspection. In *Proc. of CSCW'90*, pages 169 – 181, 1990.

[7] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, 1990.

[8] J. Conklin and M. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Trans. on Office Information Systems*, 6(4):303–331, 1988.

[9] B. Curtis. Implication from Empirical Studies of the Software Design Process. In *Proc. of Int. Conf. by IPSJ to Commemorate the 30th Anniversary.*

[10] T. DeMarco. *Structured Analysis and System Specification.* Yourdon Press, 1978.

[11] P Dewan and J. Riedl. Toward Computer-Supported Concurrent Software Engineering. *IEEE Computer*, 26(1):17–26, 1993.

[12] C. Ellis and J. Wainer. A Conceptual Model of Groupware. In *Proc. of CSCW'94*, pages 79–88, 1994.

[13] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware : Some Issues and Experiences. *Commun. ACM*, 34(1):38–58, 1991.

[14] O. Gotel and A. Finkelstein. Contribution Structures. In *Proc. of Second IEEE International Symposium on Requirements Engineering*, pages 100–107, 1995.

[15] F. Harmsen and S. Brinkkemper. Computer Aided Method Engineering. In *Proc. of the 4th Workshop on the Next Generation of CASE Tools*, pages 125–140, 1993.

[16] M.A. Jackson. *System Development.* Prentice-Hall, 1983.

[17] G. E. Kaiser. Rule-Based Modeling of the Software Development Process. In *Proc. of the 4th International Software Process Workshop*, 1988.

[18] H. Kaiya, M. Saeki, and K. Ochimizu. Design of a hyper media tool to support requirements elicitation meetings. In *Proc. of IEEE Seventh International Workshop on Computer-Aided Software Engineering*, 1995.

[19] K. Lai, T. W. Malone, and K. Yu. Object Lens : A Spreadsheet for Cooperative Work. *ACM Trans. on Office Information Systems*, 6(4):332–353, 1988.

[20] J. Lee. Extending the Potts and Bruns Model for Recording Design Rationale. In *Proc. of 13th ICSE*, pages 114–125, 1991.

[21] T.G. Lewis. *CASE : Computer-Aided Software Engineering.* Van Nostrand Reinhold, 1991.

[22] A. MacLean and R. Young. Questions, Options and Criteria : Elements of Design Space Analysis. *Human-Computer Interaction*, 6(3 & 4):201–250, 1991.

[23] B. Magnusson, U. Asklund, and S. Minör. Collaborative Document Production Using Quilt. In *Proc. of CSCW'88*, pages 206–215, 1988.

[24] V. Mashayeki, J. Drake, W. Tsai, and J. Riedl. Distributed, Collaborative Software Inspection. *IEEE Software*, 10(5):66–75, 1993.

[25] M. Muller and S. Kuhn. Taxonomy of PD Practices : A Brief Practitioner's Guide. *CACM*, 36(4):24–29, 1992.

[26] K. Narayanaswamy and N. Goldman. "Lazy" Consistency : A Basis for Cooperative Software Development. In *Proc. of CSCW'92*, pages 257 – 264, 1992.

[27] B. Nuseibeh, J. Kramer, and F. Finkelstein. Expressing the Relationships between Multiple Views in Requirements Specification. In *Proc. of the 15th ICSE*, pages 187–196, 1993.

[28] G. Olson, J. Olson, M. Carter, and M. Storr≤sten. Small Group Design Meetings : An Analysis of Collaboration. *Huma-Computer Interaction*, 7(4):347–374, 1992.

[29] B. Peuschel and W. Schafer. Concepts and Implementation of a Rule-based Process Engine. In *Proc. of 14 th ICSE*, pages 262–279, 1992.

[30] C. Potts, K. Takahashi, and A. Anton. Inquiry-Based Requirements Analysis. *IEEE Software*, 11(2):21–32, 1994.

[31] W. Robinson. Negotiation Behavior During Requirement Specification. In *Proc. of 12th ICSE*, pages 268 – 276, 1990.

[32] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lonrensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

[33] M. Saeki, K. Iguchi, K. Wen-yin, and M. Shinohara. A Meta-Model for Representing Software Specification & Design Methods. In *Information System Development Process*, pages 149–166. North-Holland, 1993.

[34] M. Saeki, K. Matsumura, and J. Shimoda. Structuring Utterance Records of Requirements Elicitation Meetings Based on Speech Act Theory. Technical report, Tokyo Institute of Technology, 1995.

[35] M. Saeki, S. Sureerat, and K. Yoshida. Supporting Distributed Individual Work in Cooperative Specification Development. In *Lecture Notes in Computer Science (CISMOD'95) to be appeared*, 1995.

[36] J.R. Searle. *Speech Acts : An Essay in the Philosophy of Language.* Cambridge Univ. Press, 1969.

[37] S. Shlaer and S.J. Mellor. An Object-Oriented Approach to Domain Analysis. *ACM SIGSOFT Software Engineering Notes*, 14(5):66–77, 1989.

[38] D. Walz, J. Elam, and B. Curtis. Inside A Software Design Team: Knowledge Acquisition, Sharing, and Integration. *CACM*, 36(10):63–77, 1993.

[39] T. Winograd. Where the action is. *BYTE*, 13(13):256–260, 1988.

[40] E. Yourdon and L.L Constantine. *Structured Design : Fundamentals of a Discipline of Computer Program and Systems Design.* Prentice-Hall, 1979.