Distributed Cognition in Software Engineering

A Mapping Study

Mathieu Lavallée, Pierre N. Robillard, Samuel Paul
Département de génie informatique et logiciel
Polytechnique Montréal
Montreal, Canada
{mathieu.lavallee, pierre.robillard, samuel.s-paul}@polymtl.ca

***Abstract*—This paper presents a mapping of the current research in distributed cognition in software engineering, using the systematic literature review approach. The result of the review shows that the literature focuses on the situational awareness of the software development team, mostly through the identification of team experts and the dissemination of task details. Research on cognitive support tools are mostly speculative, with little validation of the recommendations provided. Research on the impact of spatial disposition on team cognition is emerging, along with research on the impacts of certain emotional states. Very few papers are however concerned on the impacts of project, process and organizational constraints on team problem solving.**

Keywords; Distributed cognition; software development team; literature review; team meta-cognition; team situation awareness; team problem solving

I. Introduction

The concept of distributed cognition was first introduced in 1995 by Edwin Hutchins et al. [1] to explain how an individual can resolve problems through means beyond his internal cognitive processes. Distributed cognition observes how problems are resolved through the cognitive system around one or more minds.

The observation of distributed cognition can be applied to one individual in his/her environment. In that case, the researcher observes how the person interacts with tools around him (work documents, written notes, software, etc.).

Distributed cognition becomes especially interesting when applied to the study of teamwork. The observation of distributed cognition in team settings shows how information is transferred within the team and how solutions are created, judged and transformed by teammates.

The objective of this mapping study is to categorize the main answers given by the literature, along with potentially interesting future research avenues.

The selection process used for the literature review is presented in Section II. Section III presents an overview of the selected papers. Section IV presents a discussion of the conclusions of the selected papers as they relate to the concept of distributed cognition. Finally, Section V presents the overall conclusions of the review and introduces future research avenues.

II. Methodology

As a mapping study [4], this review is based on a lightweight version of the systematic literature review process described in the works of Barbara Kitchenham et al. [2, 3]. This section describes how the databases were searched in order to find the relevant papers and the criteria used for the paper selection and finally how the mapping and the conclusions were obtained.

A. Databases and Search String

The objective of the search is to find the published papers relevant to the subject of distributed cognition research in software engineering. The search was limited to the "Compendex" and "Inspec" databases of the "Engineering Village". The resulting search string, shown in Figure 1, returned 171 papers.

```
("software development" OR "development process" OR "software design" OR "software process" OR "software implementation")
AND
("distributed knowledge" OR "collaborative decision" OR "distributed decision" OR "distributed cognition" OR "collaborative problem solving" OR "collaborative knowledge" OR "team knowledge" OR "distributed problem solving" OR "team cognition" OR "team decision" OR "team understanding" OR "team problem solving" OR "collaborative understanding")
```

Figure 1. Final search string.

B. Selection Process

The selection process adds three more steps to the initial search, which are based on the title, the abstract, and the full text. The selection from the titles is limited to the removal of duplicate papers and conference proceedings introductions. The selection from the abstracts kept papers containing both software and cognition concepts in their abstracts. The selection from the full text removed low quality papers. The quality was evaluated through the identification of context descriptions and data collection methodologies. Papers without these elements were removed. Some theoretical papers were kept, based on the apparent validity of the model presented.

The selection process retained 24 papers. The documents produced by the process are available on request.

C. *Data Synthesis Process*

To perform an accurate synthesis based on our extracted data, we need to manage various types of qualitative data. From the thirteen synthesis approaches described by Cruzes and Dyba [5], we chose the Grounded Theory approach, because it is designed to work with a wide spectrum of qualitative data. We limited ourselves to the three following steps from the Anselm L. Strauss [6] works:

- Associate one keyword to each extracted conclusion,
- Regroup the keywords into concepts,
- Describe how the conclusions complete or contradict each other.

For a more thorough description of the application of Grounded Theory to the software engineering domain, the reader is invited to read the works of O'Connor et al [7, 8] and Lavallée et al [9].

III. Results

This section present the results of the mapping study, where the selected papers are identified by the letter 'S', as described in Appendix A.

A. *Study Methodology*

Table I shows that most of the selected papers describe empirical and academic research, with a single paper whose context is labeled "Open".

Industrial context studies describe real software development projects performed in professional organizations. Academic context observes the work of students performing a formative task. The open context refers to a study performed on an open source development project. This open source community can include both professionals and academics. Note that some papers are purely theoretical and therefore do not present any study context.

Table II presents the many approaches used for data collection in the various selected papers. Some papers used multiple data collection approaches, therefore the total does not add up to 24.

TABLE I. Research Context of the Selected Papers

Context	#	Papers
Industrial	17	[S1], [S2], [S3], [S4], [S5], [S9], [S10], [S13], [S16], [S18], [S20], [S21], [S23], [S24].
Academic	10	[S8], [S11], [S12], [S15], [S17], [S19].
Open	1	[S22]

The survey questionnaire is mainly used to confirm or refute the conclusions obtained with other types of data, although some papers base their conclusions on the survey questionnaire alone. The artifact evaluation consists in the analysis of documentation issues of the software development process. This evaluation is often used to evaluate the quality of the work done, and thus the performance of the team. The semi-structured interview describes face-to-face meetings, which is often used to obtain feedback from the software developers. The non-participatory observation occurs when researchers observe the work done by software developers without interfering directly with them, a technique called "shadowing". The audio-video approach consists in the recording of work sessions performed by the software development team. These recordings can include conversations between team partners, computer screen capture videos, keystroke logging records, etc. Usage data consists in statistical measurements obtained from the use of specific software tools. These measurements show the usage frequency of the different functionalities available. These data help researchers in understanding how the software developers adapt software tools to their tasks. Participatory observation occurs when the researcher actively participates in the observed task. This approach enables a more accurate recording of the internal cognitive processes required to perform the task, at the cost of a significant bias.

TABLE II. Data Collection Methodology of the Selected Papers

Approach	#	Papers
Survey questionnaire	11	[S8], [S11], [S12], [S17], [S18], [S22], [S24].
Artefact evaluation	10	[S1], [S2], [S3], [S5], [S8], [S11], [S12], [S13], [S15].
Semi-structured interview	7	[S3], [S5], [S16], [S19], [S20], [S24].
Non-participatory observation	6	[S3], [S5], [S9], [S13], [S24].
Audio-video	3	[S1], [S2], [S9].
Usage data	3	[S21], [S23].
Participatory observation	2	[S4].

IV. Discussion

This section presents the conclusions of the selected papers, based on the concepts found through the Grounded Theory approach.

The results of the synthesis can be related to Vygotsky's triangular model of mediated interaction [10], which stated that the activities performed by the software developers within their teams are always mediated by their environment. As Engestrom [11] elaborates, this mediation can take the first four forms presented in Table III.

The new "Emotion" form was motivated by the multiple studies evaluating the emotional state of the team. The importance of emotions during problem-solving has been deemed critical by recent research. Damasio insist on the fact that *"the presumed opposition between emotion and reason is no longer accepted without question"* [12]. Emotion must be considered alongside cognition.

A. *Community: The Software Development Team*

Distributed cognition in software engineering is closely related to how the team assists the individual developer. To be an effective mediator, the team must have coherent situational awareness. Situational awareness is defined as the knowledge a person has of himself/herself and his/her surroundings [13]. In software engineering research, this awareness is oriented along two axes: team meta-cognition and task awareness.

TABLE III. FORMS OF MEDIATIONS

Form	Description	In software engineering
Community	Team interaction	Development team
Instruments	Tools and artefacts used	Individual tools, groupware.
Division of Labor	Tasks performed	Team topology and team structure.
Rules	Impact of disciplined	Process, project, organization
Emotion	Emergent state	Motivation

Team meta-cognition is defined as what each team member knows on the knowledge of their teammates. Good team awareness implies that team members know who the experts are and who are reliable sources of information. It is related to the "who knows what". Task awareness is related to the team's shared mental model of the work to do. The more this mental model is coherent between team members, the team's environment and the relevant stakeholders, the better is task awareness.

1) Team Meta-Cognition

The importance of team meta-cognition has been outlined by the works of Kraut and Streeter [14], cited by [S7]:

"Experimentation has shown that developers valued other people as their most used source of help when developing software."

This observation has also been reported by Glor and Hutchins [S1]. When one team member is stuck on a problem, he can present it to one of his partner in order to start a discussion on the most appropriate solution.

The best source of information for software developers is their teammates. It is therefore important for the developer to know who hold this information within the team. This becomes problematic when the team meta-cognition is weak: Sub-optimal choices can be made because the decision-makers are not aware that better solutions exist. Similarly, team performance can be affected when the identified sources of information are not appropriate. Walz et al. [S2] report a case where the two developers with the most influence on decision-making where the ones with the less experience. The team had a poor perception of its own knowledge because the appropriate experts were not identified, resulting in a poor choice of solutions.

To resolve meta-cognition problems, many studies present specific methods [S8, S10, S11, S14, S16, S17]. For example, Kettunen [S10] recommends the identification of "knowledge dependencies" within the team. He presents the importance of information change propagation within the team: A developer must be aware of the people around him capable of providing information changes relevant to his work.

Ye's paper [S14] recommend the identification of expert related to the number and size of modifications made in a code module. Such a tool could enable a developer to contact directly the person most susceptible to know how this code works.

Hause et al. [S8] demonstrate the importance of efficient communications. Their research shows that high performance teams communicate *less* than lower performing ones, because their exchanges are better targeted and better structured. A better knowledge of who are the experts within the team could, for example, limit the communication exchanges to the person most susceptible to provide a relevant answer. Their conclusion [S8] is confirmed by Espinosa et al. [S16]. The later shows that a software development team distributed on distant sites possesses a better knowledge of its own experts. The difficulty of exchanging information over distant sites forces developers to have a better knowledge of the reliable information sources. Sarker et al. [S11] paper shows that sources providing large amounts of accurate information have the greatest impact on knowledge transfer.

Finally, He et al. [S17] show that team meta-cognition is essentially a matter of time. Their paper presents a significant correlation between the self-evaluation of the performance of the teams and the quality of the software product as the project progresses. The impact of familiarity between team partners, initially very strong, diminishes as the team members learn to know themselves better. The team has therefore a better vision of the strengths and weaknesses of their partners, and thus obtains a better self-evaluation of their performance.

2) Task Awareness

Better task awareness is mostly useful for the planning and coordination of the work. As Espinosa et al. [S16] explain, a shared knowledge of the task helps team coordination. For example, the use of a public media like the wall board of Sharp et al. [S13] improves team coordination by publicizing immediately any change in the state of the cognitive system. This immediate propagation of changes enables better team situational awareness. This immediate propagation also ensures that the mental model of the task remains synchronized throughout the team, as shows Kettunen [S10]. The presence of a synchronized mental model also diminishes the need to communicate, and thus improve the performance of the team Hause et al. [S8].

De-Franco Tomarello [S12] also shows that if an initial model of the task is imposed upon the team, it improves the problem comprehension. An initial model enables the team to start with a shared mental model better structured and a better organized.

The works of Flor and Hutchins [S1] and Spinuzzi [S5] outline the adaptation of the information received to the context of the task. They show that developers reuse and adapt the information obtained according to their immediate needs. Spinuzzi adds that the artifacts given to the developers are not used in the manner planned, but they are rather adapted to the nature of the task. Information must therefore be designed to be compatible to the needs of the task. Spinuzzi notes that important information resources are ignored because their usability in the context of the task is weak. Developers have therefore diminished task awareness because they do not have all the relevant information in hand. Spinuzzi's concerns are confirmed by Conradi and Dingsoyr [S4], who warn that inadequate data repositories become data cemeteries.

B. *Instruments: Cognitive Support Tools*

The mediating instruments are the various artifacts and tools used by the developers. Among the many tools available, some have an explicit objective to support individual and group cognitive tasks. Cognitive support tool research in software engineering is oriented along two axes: Tool supporting individual cognition, and tool supporting team cognition ("groupware").

1) *Individual Cognition Support Tools*

The papers on individual cognition present two tools common in software development environments: code completion [S14] and compilers' error list [S9].

Code completion, presented by Ye as a cognitive support tool [S14], is a feature of most modern integrated development environments (IDE). This tool recall to the developers all the words understood by the compiler. This enables them to speed up their works by giving them context-aware information. Given the large size and complexity of software components this tool is an essential asset of the software developer.

Walenstein [S9] shows that compilers' error list assists developers in their debug planning by providing a list of the problems found with links to the relevant code snippet. This tool facilitates the developer's work, who only needs to identify the reason for the problem, and not where the problem is located.

2) *Team Cognition Support Tools*

Groupware tools contains the management of public communication channels like wall boards, wiki software, shared calendars, web forums and audio-videoconference tools [S13]. The main characteristic of these tools is that they are transparent as to the origin and destination of the information transmitted. The drawback of this is that users of the system have access to data which do not concern them. In one specific case, a discussion forum had to be moved from a public to a private space, because it created exaggerated expectations from some of its users [S21]. However, private communication channels are also essential for efficient information exchange. Software developers can exchange intermediate steps of a work-in-progress to a team partner without concern for public judgment [S24].

Many papers on team cognition support present the required functionalities for collaborative tools ("groupware"). For example, De Franco-Tomarello et al [S7] list the following key functionalities to ensure that groupware offer a support for collective decision-making, team situation awareness, and sharing mental models :

- Ability to support the team communication channel,
- Ability to support the team collective tools, like planning tools, design tools and knowledge bases,
- Ability to support a collaborative approach to modeling.

Walz et al. [S2] add the necessity to document the rationale behind the choices made by the team. They blame current groupware solutions which report information without reporting how the information was obtained. They argue that the decisions made must be documented with more details.

Whittaker and Schwarz [S3] compare the advantages of a planning tool like Microsoft Project to a *kanban*-style wall board of tasks. They show that the wall board is beneficial because of its public aspect and its flexibility. They argue that current groupware are too restrictive and do not enable different planning approaches. They show however that the wall board is difficult to transmit to stakeholders on distant sites, and that it is difficult to make major changes. It is also not possible to follow the version changes on the wall board, contrarily to a software tool. Finally, it is not possible to present different views of the data when using the wall board.

Research on groupware took a different turn with the emergence of the "Web 2.0". A software team can now cook up a collaborative framework of tools from a plethora of tools available on the Cloud. For example, a team can use a knowledge base managed with Drupal (www.drupal.org), track its development issues with Bugzilla (www.bugzilla.org), plan their tasks with Trac (www.trac.edgewall.org), and keep contact with each others with Pidgin (www.pidgin.im).

C. *Division of Labor: The Structure of the Team*

The division of labor mediator describes the actual tasks performed by the different members of the team. It also considers how the team is spatially disposed, as the physical workspace can have an important impact on the interactions taking place.

For example, it is important to plan the disposition of team members and their communication channels when the team is distributed. The theoretical model of Kubasa and Heiss [S6] proposes and optimization of information flows based on geographical distances, hierarchies, cultural difference and personal familiarity (friendship, rivalry). This model also enables the calculation of a communication cost and of the probability of a delivered message without error.

Meneely and Williams [S23] focused instead on the modeling of a real case; a software development forum. Through a statistical analysis of its usage data, they identified the people performing the roles of "solution providers" and "solution approvers". They noted that approvers, those who choose a solution and implement it in code, are central in the communication network of the forum. Their statistical approach enables an evaluation of the state of an open-source community.

Bass et al. [S20] recommends that the physical disposition of team members across distant sites must consider team meta-cognition. The identification of domain experts during team construction ensures that every developer knows the reliable sources of information (see section IV.A.1). They also say that it is important that each distant site has one person acting as developer in the team.

This ensures a proper dissemination of information across the multiple sites despite the distance.

D. Rules: Project, Process and Organizational Constraints

Lavallee et al. [15] work on the impacts of processes on individual developers concluded that the impacts are not often considered despite having serious detrimental effects. The conclusions are similar at the team level: The impacts of mediating rules on the software development team are rarely observed. Stubblefield and Carson [S18] outline this concern by urging managers not to impose strict rules on the use of a groupware tool: Usage must be adapted to the cognitive needs of the task, and not the other way around.

Hause et al. [S8] note that decision-making mechanisms can be different from one team to another. As Falessi et al. [S15] note, having different process at the individual level is not problematic, but it can become critical at the team level. They show that to impose a decision-making approach with explicit alternative research improve the decision quality from 11% to 67%.

However, decision-making cannot be delayed indefinitely. As Walz et al. [S2] show, software development projects are split into two phases: A decision-making phase and an execution phase. The acquisition of new knowledge must occur at the beginning of the project; if this information arrives only after the midpoint of the project, it is typically ignored. Walz et al. report that adding experts after this knowledge acquisition phase has no impact on the decisions made beforehand. The development team has already made its decisions and does not want to roll back.

However, this capacity to roll back decisions is one characteristics of good working teams, as observed by Hause et al. [S8]. Good teams, having a better shared mental model of the work to do and a better knowledge of the experts in their midst, make less decisions than other teams, but are more ready to roll back and change previous decisions. Good teams changed 20% of their decisions, against only 9% for the bad teams.

E. Emotion: Team Emotional States and Motivation

One of the aspects uncovered by research in cognitive psychology is the fact that individual performance changes when the person's emotional state changes. We can observe the same fact at the team level, and thus the emotional state of the team can also affect its performance. Marks and Mathieu describes these emotional states as "emergent states":

"Emergent states describe cognitive, motivational, and affective states of teams, as opposed to the nature of their member interaction." [16]

For example, team topology can stay stable for the duration of the project, but the emotional state can change within a single day, or even a single meeting. Whittaker and Schwarz [S3] studies the impact of a material wall board of tasks on the team sense of belonging. This wall board requires developers to cut pieces of paper detailing their estimations and to stick it to the wall. The manual aspect and the public nature of the board improve the perceived responsibility of the developers toward their task estimations. There is a certain shame in having to correct the content of the wall board; therefore the estimations are more carefully made. By contrast, the software tool is very often "write-only": Developers enter data in the tool, but they never read it. The quality of the estimations in the software tool is much weaker.

Parsarnphanich and Wagner [S22] study the motivation of important contributors to the Wikipedia knowledge base. They show that the greatest motivator to contribution is the quick feedback they receive from their contributions, even when the change is minor. This quick feedback outlines the public aspect of their contribution and incites them to continue. These two papers show that pride can affect performance [S3] and productivity [S22]. They also show that pride can be controlled by the public aspect of the task.

Trust is also an important emotion for certain team cognition elements, like meta-cognition. An initial face-to-face meeting seems to have an important impact on what the developers perceived of their supervisors. Bass et al. [S20] show that a visit from the manager to all the distant sites can improve communication for the project duration. Richardson et al. [S19] note that if the supervisors did not meet the other team members face-to-face, the developers did not ask them questions.

Motivation is also an essential emotion required for the success of any project. Wikis and other collective memory knowledge bases are dependent on the volunteer work of motivated individuals. A pragmatic altruism or idealism has been identified has a major factor of Wikipedia's success [S22]. Managers must encourage a cooperation culture within their teams. Tools must be able to support the implication of the team partners by promoting a good compatibility with the work to perform [S5].

V. CONCLUSIONS

The main conclusion from the synthesis is that there is no consensus on how to manage distributed cognition in software development: Many papers describe what should be done, but very few describe how to do it. The use of varied practices in varied contexts means that comparison between studies is very difficult, since contradictory conclusions abound.

There are also many suggestions for new functionalities for cognitive-support software tools. There are therefore few empirical studies of the software functionalities considered as important. One study notes that the main weakness of the existing groupware tools is that they lack flexibility [S3]: Software developers prefer using many tools more adapted to their task rather than one generic all-purpose tool.

Additionally, there are very few studies on the ergonomics of software tools. The papers describe what collaborative tools ("groupware") must support, but they do not describe how this support can be ensured. There is therefore not enough research on the affordance of collaborative tools available for software development teams.

There are also no papers on the mental workload of software developers. We do not know the impact of collaborative tools on the mental workload of software developers. There is no information on the global mental workload of the software team, nor whether cognitive effort is appropriately spread across team partners.

ACKNOWLEDGMENT

We thank Professor Jean-Marc Robert for the initial idea of a literature review on the subject of distributed cognition.

REFERENCES

[1] E. Hutchins, Cognition in the wild. Cambridge, MA: MIT Press, 1995.

[2] B. Kitchenham, "Procedures for Performing Systematic Reviews," 2004.

[3] J. Biolchini, P. Gomes Mian, A. Candida Cruz Natali, and G. Horta Travassos, "Systematic Review in Software Engineering," Rio de Janeiro, 2005.

[4] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using Mapping Studies in Software Engineering," in Proceedings of Psychology of Programming Interest Group, 2008, vol. 2, pp. 195–204.

[5] D. S. Cruzes and T. Dybå, "Research synthesis in software engineering: A tertiary study," Information and Software Technology, vol. 53, no. 5, pp. 440–455, May 2011.

[6] A. L. Strauss, Qualitative Analysis for Social Scientists. Cambridge, UK: Cambridge University Press, 1987, p. 319.

[7] G. Coleman and R. O'Connor, "Software Process in Practice: A Grounded Theory of the Irish Software Industry," in Software Process Improvement, vol. 4257, I. Springer Berlin / Heidelberg, 2006, pp. 28–39.

[8] S. Basri and R. V. O'Connor, "Understanding the Perception of Very Small Software Companies towards the Adoption of Process Standards," in Systems, Software and Services Process Improvement, vol. 99, Springer Berlin Heidelberg, 2010, pp. 153–164.

[9] M. Lavallée and P. N. Robillard, "The Impacts of Software Process Improvement on Developers: A Systematic Review," in 34th Intl Conf on Software Eng (ICSE 2012), 2012.

[10] L. S. Vygotsky, Mind in Society: The Development of Higher Psychological Processes. Cambridge University Press, 1978.

[11] Y. Engeström, "Activity theory as a framework for analyzing and redesigning work.," Ergonomics, vol. 43, no. 7, pp. 960–74, Jul. 2000.

[12] A. Damasio, The Feeling of What Happens: Body and Emotion in the Making of Consciousness. Houghton Mifflin Harcourt, 2000, p. 400.

[13] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems," Human Factors, 37 (1), pp. 32–64, 1995.

[14] R. Kraut and L. Streeter, "Coordination in Software Development," Communications of the ACM, 38, no. 3, 1995.

[15] M. Lavallée and P. N. Robillard, "The Impacts of Software Process Improvement on Developers: A Systematic Review," in International Conference on Software Engineering, 2012.

[16] M. A. Marks, J. E. Mathieu, and S. J. Zaccaro, "A temporally based framework and taxonomy of team processes," Academy of Management Review, vol. 26, no. 3, pp. 356–376, 2001.

APPENDIX A. THE SELECTED PAPERS

#	Reference
S1	N.V. Flor and E.L. Hutchins, (1991). *Analyzing distributed cognition in software teams: a case study of team programming during perfective software maintenance*. Empir. Stud. of Program.: 4th Workshop.
S2	D. Walz et al., (1993). *Inside a software design team: Knowledge acquisition, sharing, and integration*. Comm. of the ACM.
S3	S. Whittaker and H. Schwarz, (1999). *Meetings of the board: the impact of scheduling medium on long term group coordination in software development*. Computer Supported Cooperative Work.
S4	R. Conradi and T. Dingsoyr, (2000). *Software experience bases: a consolidated evaluation and status report*. PROFES 2000.
S5	C. Spinuzzi, (2001). *Software development as mediated activity: Applying three analytical frameworks for studying compound mediation*. ACM SIGDOC Intl Conf. on Comp. D.
S6	G. Kubasa and M. Heiss, (2002). *Distributed face-to-face communication in bottom-up driven technology management - A model for optimizing communication topologies*. IEEE Intl Eng. Mngmnt Conf.
S7	J. DeFranco-Tommarello and F.P. Deek, (2002). *Collaborative software development: a discussion of problem solving models and groupware technologies*. 35th Hawaii Intl Conf. Syst & Science.
S8	M. Hause et al., (2003). *Performance in international computer science collaboration between distributed student teams*. 33rd Annual Frontiers in Education.
S9	A. Walenstein, (2003). *Observing and measuring cognitive support: steps toward systematic tool evaluation and engineering*. 11th IEEE Intl Workshop on Program Compreh.
S10	P. Kettunen, (2003). *Managing embedded software project team knowledge*. IEE Proceedings: Software.
S11	Sa. Sarker et al., (2003). *Knowledge transfer in virtual information systems development teams: an empirical examination of key enablers*. 36th Hawaii Intl Conf. on Systems Sciences.
S12	J. DeFranco-Tommarello, (2003). *A study of collaborative software development using groupware tools*. Proceedings. ITRE.
S13	H. Sharp et al., (2006). *The role of story cards and the wall in XP teams: a distributed cognition perspective*. AGILE .
S14	Y. Ye, (2006). *Supporting software development as knowledge-intensive and collaborative activity*. ICSE 2006.
S15	D. Falessi et al., (2006). *Documenting design decision rationale to improve individual and team design decision making: An experimental evaluation*. ISESE'06.
S16	J.A. Espinosa et al., (2007). *Team knowledge and coordination in geographically distributed software development*. J. Mngmnt Inf. Syst.
S17	J. He et al., (2007). *Team cognition: Development and evolution in software project teams*. J. Mngmnt Inf. Syst.
S18	W.A. Stubblefield and T.L. Carson, (2007). *Software design and engineering as a social process*. Conf. on Human Factors in Computing Systems.
S19	I. Richardson, S. Moore, D. Paulish, V. Casey and D. Zage, (2007). *Globalizing software development in the local classroom*. Software Engineering Education Conf.
S20	M. Bass et al., (2007). *Collaboration in global software projects at siemens: An experience report*. ICGSE.
S21	P.-H. Cheng et al., (2008). *collaborative knowledge management process for implementing healthcare enterprise information systems*. IEICE Trans. on Inf. and Syst.
S22	P. Prasarnphanich and C. Wagner, (2009). *The role of wiki technology and altruism in collaborative knowledge creation*. J. Comput. Inf. Syst.
S23	A. Meneely and L. Williams, (2011). *On the Use of Issue Tracking Annotations for Improving Developer Activity Metrics*. Adv. Softw. Eng.
S24	S. Patil et al., (2011). *Methodological reflections on a field study of a globally distributed software project*. Inf. Soft.Tech.