Dams, Flows and Views: Cross-Aspect Use of Knowledge in Collaborative Software Development

Norbert Jastroch
MET Communications
Eschbacher Weg 10
61352 Bad Homburg, Germany
norbert.jastroch@metcommunications.de

Vassilka Kirova
Alcatel-Lucent
600-700 Mountain Ave.
Murray Hill, NJ 07974
vassilka.kirova@alcatel-lucent.com

Thomas Marlowe
Seton Hall University
400 W. South Orange Ave.
South Orange NJ 07079
thomas.marlowe@shu.edu

Mojgan Mohtashami
Advanced Infrastructure Design
10 Richardson Lane
Hightstown, NJ 08520
mojgan@aidpe.com

Abstract

Collaboration between organizations raises significant knowledge management issues, especially in software development of complex projects, in which both product and process are themselves knowledge. While research has examined direct, explicit flows of knowledge within project aspects, or forward between aspects, there is less investigation of the need and support for backward, implicit or emergent flows.

Keywords: Collaborative software development, collaboration, software engineering, knowledge management, ICSD.

1 Introduction

The share and impact of inter-organizational collaborative software development (ICSD), in various modes [19] and with multiple motivations have increased. Concurrent trends of growing complexity, feature space and size of software packages, which are also increasingly knowledge intensive, are characteristic of the majority of projects. Many of these applications can be expected to be long-lived, evolvable, and used in diverse contexts and environments. This combination of factors entails use of sophisticated and specialized organizational, software engineering, and knowledge management (KM) approaches. We consider a software development project *hard* if it is large, complex, and knowledge-intensive, and intended to be long-lived, evolvable, portable, and useful in diverse settings or for diverse user populations or clients.

Collaboration in general, and collaborative software development for hard projects in particular, requires cooperation, information sharing, and interaction at multiple levels. Working more-or-less from the governance business aspects toward the technical and deployment ones, and forward in project time, we identify in Section 2 a number of critical, knowledge-intensive aspects of collaborative software development, particularly crossing organizational boundaries.

In past papers , we and others have investigated the impact of collaboration in hard projects, and recommended changes in policies, processes and artifacts. These papers have addressed both general concerns [4,9,19,22,24,25] and specific areas such as business policies and processes [10,15], risk management [16,17], and technical processes and artifacts [11,12,13,23].

These recommendations affect corporate policy and procedures, software development, risk management, and knowledge management. Major themes are (1) a layered approach, comprising single-organization structures, a collaborative structure, and a method of resolving priorities and conflicts; and (2) methods and/or artifacts to extract, communicate and display appropriate knowledge, possibly including new kinds and forms of information, as well as filters, abstractions and views.

In the KM literature [2.6,7,8,14], knowledge is frequently classified as explicit, implicit or tacit; it may also be useful to distinguish emergent knowledge—knowledge that arises from synthesis of existing knowledge, or is a result, possibly in combination with such knowledge, of the project or product under investigation. Collaborative knowledge (see [5,7,14]), particularly the more difficult to control tacit and/or emergent, poses its own problems, most particularly those of intellectual property, security, privacy, and confidentiality, on the one hand, and credit (cost-benefit) assignment on the other [14]. With care, it is not that difficult to create a structure for the sharing and use of such knowledge, especially if used within an aspect, or when the flow is forward, that is, used as a driver of tasks more immediately focused on the current project, process, or product. It is more difficult when the flow itself is implicit/tacit or emergent, and especially if the flow is backward, that is, from a more product focused back to a more process or policy focused context.

In Section 3 we review and extend a list of drivers, benefits, impediments and risks in collaborative software development for hard projects, thus identifying the dams. Section 4 presents some examples of emergent and backward flows and related views.

The final Section 5 briefly presents recommendations and conclusions.

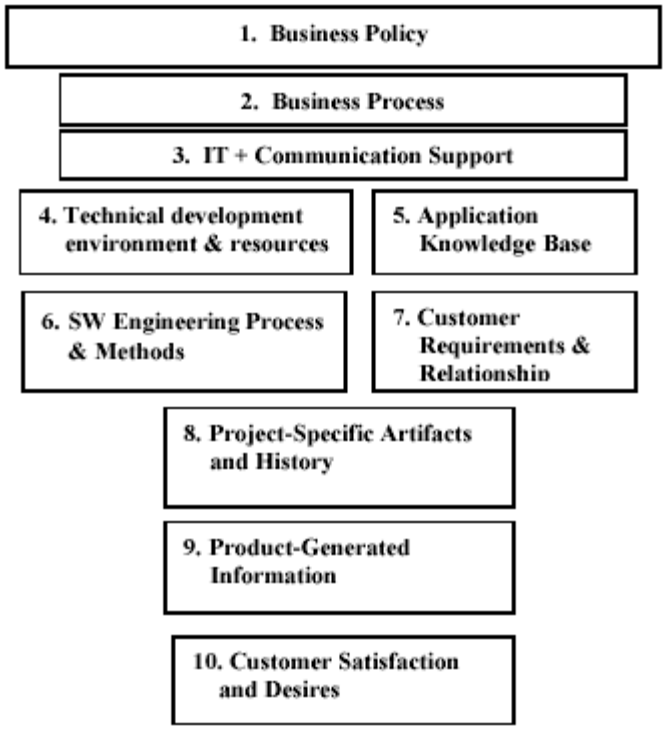
2 Aspects of software development

Collaboration in general, and collaborative software development for hard projects in particular, requires cooperation, information sharing, and interaction at multiple levels. Working more-or-less from outside in (governance and business drivers to development and domain platform to specific project and product), and forward in time, we can identify a number of critical, knowledge-intensive aspects of collaborative software development, particularly crossing organizational boundaries.

1. Business policy: Includes business vision and plans, risk tolerance, legal (intellectual property, proprietary information, privacy, confidentiality, and related issues), collaboration readiness and advocacy, marketing and management strategies, and issues related to reputation, business culture, and openness to employees, collaborators and customers.
2. Business process: Includes security, risk management and knowledge management, personnel management (including attitude toward collaborative work), culture and trust, marketing, and support for extramural activities.
3. IT and related support: Communication infrastructure and restrictions, establishment of shared representations and glossaries (see [15]).
4. Application knowledge base: Domain (e.g., banking) and product discipline and functions (e.g., auditing) knowledge. Heterogeneous contributions of partners; integration and inclusion of external knowledge, including new developments; supporting extramural use; credit and debit assignment; support of domain expert/discipline specialist consultation and collaboration [3].
5. Technical development environment and resources: Development platforms: computing resources; software tools including change management and dependency tracking.
6. Software engineering process and methods: Includes technical management processes including requirements analysis and quality assurance; people issues such as training and team management; nature of artifacts to be developed in SW process, and patterns of use, dependence and sequencing of these artifacts. Requirements for documentation and views.
7. Customer requirements and intimacy. Initial and ongoing interaction with customer (and possibly other stakeholders), prior to release, or explicit requests for modifications.
8. Project and product artifacts and history: Includes definition and design time software artifacts and change history. The actual artifacts associated with the current project and/or product: Requirements, specification, architecture, design, code, documentation, dependence analysis and traceability, testing and debugging. Interacts with Customer Requirements.
9. Product-generated information: Information resulting from use and/or analysis of product: input-output patterns, including unexpected exceptions or errors, patterns of use and performance based on information from profilers, history, logs, and similar tools, results of static and dynamic compiler analyses and transformations,
10. Customer satisfaction and desires: customer satisfaction survey results, modification requests and theirs severity and scope, ongoing feedback, new feature requests and long-term partnering proposals

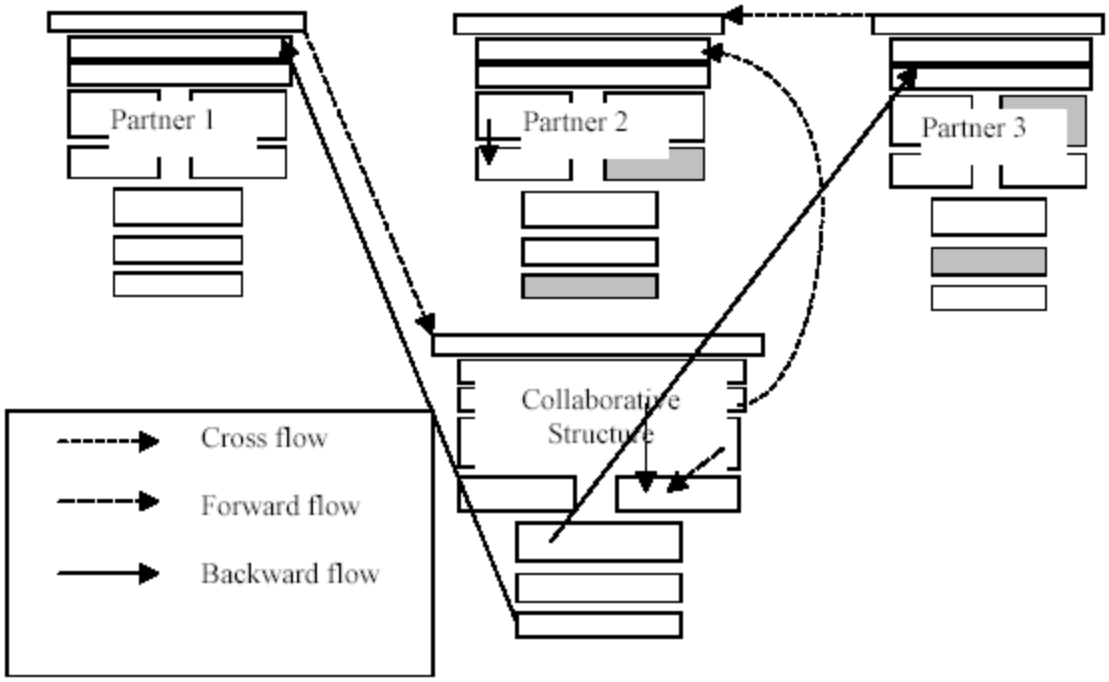
Each aspect generates and ideally consumes its own information, and must manage that information for efficient use. Each aspect may but need not exist for each partner and for the collaboration, and some, such as (7) and (10), will necessarily be limited to one or two collaborators as discussed in [18]. Figure 1 shows the aspect structure, and Figure 2 shows its replication in a collaborative structure/engagement. Figure 2 shows some example flows: *cross-flows* are those between identical aspects; *forward flows* are those downward in the diagram; and *backward flows* include all the others.

Figure 1. Aspects of Software Engineering



Flows out of the collaborative structure are especially likely to involve collaborative knowledge, as are those (not illustrated for reasons of simplicity) with multiple sources in multiple partners. Emergent flows are most likely to include some backward flow, and emergent knowledge is most likely to be (at least in part) carried along such flows.

Figure 2. Aspects in Collaboration



In our previous papers,we have considered modifications in both the structure in each aspect [10,11,12,13,19] and in its knowledge management to support collaboration [7,8], including supporting flows forward/downward in the process, and some of the more evident feedback flows. Here, we indicate the need for a more careful investigation of the need for additional, emergent or backward flows, to improve the collaboration, to optimize the process and product, to improve partner corporate and technical decision processes, or to improve the acquisition, organization, management, and protection of knowledge.

3 Drivers, benefits, impediments and risks in ICSD

In order to motivate the investigation of knowledge flows, we briefly review the tradeoffs in collaboration and in ICSD. These are based on existing literature and project observations, some of which have been discussed in our previous work and that of others. The identified impediments, and to a lesser extent the risks, become the dams obstructing the flow of needed information.

Drivers

1. Increase product feasibility, market, and profitability by leveraging expertise, knowledge, intellectual property, and reputation and connections of the partners.
2. Improve time to market by resource and expertise sharing, by reducing cost and time for knowledge acquisition, and training, and by parallel development.
3. Establish good working relationships with trustworthy partners.
4. Foster innovation by exploiting collaborative knowledge and collaborative process optimization.

Other Benefits

1. Increased knowledge and expertise from collaborating with specialists at other partners [3].
2. Improved tool, process and development environment, and improved component repository.
3. Better resilience due to extended personnel resource pool.
4. Improved reputation resulting from quality product and association with quality partners.
5. Innovation and insights resulting from development of knowledge and data filters, abstractions, representations and views.

Impediments

1. Corporate inertia and resistance from corporate and technical management, IT departments, and legal counsel [16].
2. Intellectual property, proprietary information, privacy, confidentiality and security.
3. Corporate policies and procedures for sharing information, firewalls, access restrictions, …
4. Difficulty in establishing trust and understanding of differences in social and corporate cultures [1,16,17,21].
5. Inconsistencies in tool suites, software development processes, and so on.

Other risks—business

1. Management contingency policies need to be collaboration-aware [18].
2. Risk management process needs to be collaboration-aware.
3. Customer and vendor contact needs to be centralized.
4. Indirect communication (e.g., via agents).

Other risks—technical

1. Specification needs to be collaboration- and decomposition-sensitive.
2. Software development process not amenable to cooperation and collaboration.
3. Inappropriate definition of component interfaces, in particular with respect to supporting evolution, both before and after release.

4 Dams, flows and views

Definitions and concerns:

- A *knowledge object* is a representation, often an abstraction, of a set of information and analysis results together with a context. The denotation, and especially the connotation, of a knowledge object is in large part defined by the domain, the discipline, the organization, and the social and organizational culture and history/memory and learning capability of an institution. One problem in collaboration lies in assuring communication not just of the object, but of enough context so that common denotations and connotations of knowledge objects can be established. Another lies in assuring that there is minimal leakage of protected information that is not needed by the recipient or the collaboration, or conversely underestimation of the cost associated with achieving minimal leakage.
- A *view* is a picture of a product, process, project, or knowledge object, arising from an angle of analyzing an object as to perceive/identify some of its aspects under given/specific interest – employs filtering, results in extraction, generates a knowledge object.
- A *flow* is a communication, with appropriate extraction, translation, filtering and abstraction, of a knowledge object available in one aspect or subaspect of a collaboration, to another aspect or subaspect in which it will be needed, or in which it will be integrated with other knowledge objects, or in which it will be further manipulated for use in a third aspect.
- A *dam* is a rule, guideline or standard related to management or technical procedures and policies, tool suites, and interfaces which, intentionally or not, regulates flows. A set of such dams works as the regulative framework for all flows in a collaboration.

The key issue in ICSD for hard projects is the tension between evolvability on the one hand, and intellectual property and related issues on the other. We have already considered modifications of management and software processes and artifacts, but largely to support later project aspects and phases, or to support change and optimization of the aspect or phase under consideration. Much of our attention has been separately focused on business structure (1)-(3), knowledge management (5), or software development (6)-(8).

A typical way to address the impediments and risks is to figure out which of these are considered controllable, establish limits/levels of acceptability for them, and implement guidelines in order to ensure that those limits be kept without unduly inhibiting progress on the project. Intellectual property issues thus can be (and frequently are) made subject to an explicit corporate policy. Customer and vendor contacts can be restricted to specified personnel, with necessary communication then being channeled through fixed reporting lines and procedures. Tool suite and process inconsistencies often get treated by general ruling in (respectively, out) of what is allowed.

In effect, once ICSD becomes a frequently used practice or even a sort of a business concept of an organization, the management of impediments and control of risks soon drive toward the introduction of guidelines or even standards for a variety of processes and technical facilities. This is normal for intra-organizational software engineering, and in this context it is usually considered to deliver a sound balance of evolvability and risk management.

However, there are clear examples of the potential utility of collaborative, emergent or backward flows, as well as the protections that may need to be applied.

New information in, or new inferences from, a partner knowledge base (5) can help in meeting customer requirements (7) or desires (10), or in improving product design (8). However, credit assignment for this information, and its use by the collaboration and by other partners remains an issue, especially when the knowledge must be integrated with knowledge available to other partners or developed by the collaboration to be useful.

Inadequacy in collaborative software engineering structures (6) may require changes in technical infrastructure (4), either for the collaboration or for individual partners, or in IT and communication support (3), or even in intellectual property policies and processes (1-2). Alternatively, the problem may be traced back to problems in sharing knowledge (5)—and perhaps again indirectly to intellectual property and security (1-3), or to inadequate development of abstractions, filters or views—a combination of (2, 3, 6).

Finally, as is well-known, information resulting from the design process (8) or the analysis or execution history of the application (9) can reveal flaws in security or confidentiality policies and processes, or be needed to tune or change risk management plans, affecting the business phases (1)-(3) and perhaps the technical infrastructure (4)-(5). But both the information and its analysis may require divulging the internals of software components or proprietary tools.

5 Conclusions

ICSD, to a far greater extent than collaboration in general, will always be driven by the tension between the overwhelming need for shared knowledge in all phases and aspects of the corporate and technical process, and the need to protect legitimate security, intellectual property, confidentiality, and privacy interests, including those of third parties not involved in the collaboration. Although the risks are real, the benefits are substantial enough to encourage greater use of this fully collaborative mode of development.

However, sharing must be guarded, by filtering and abstracting transmitted knowledge, and by providing constraint views, while still communicating the necessary information. The ubiquity of integrated and emergent knowledge, and the utility of emergent and backward flows, argue that the harder the development project, and in particular, the greater the reliance on dynamic knowledge and product evolution, the greater the anticipation of and the need for filters, abstractions and views, an agreed-on scheme for credit allocation, and an approach for mediation and conflict resolution.

6 References

[1] P. M. Alexander, Teamwork, Time, Trust and Information, Proceedings Of The 2002 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on Enablement Through Technology, Port Elizabeth, South Africa, 2002, pp 65-74.

[2] C. D. Crampton, The Mutual Knowledge Problem and Its Consequences for Dispersed Collaboration, **Organization Science**, Vol. 12, No. 3, May–June 2001, pp. 346–371.

[3] D. Flynn, E. Brown, R. Krieg: A Method for Knowledge Management and Communication within and across Multidisciplinary Teams, KGCM 2008, June 2008.

[4] J. D. Herbsleb: Global Software Engineering: The Future of Socio-technical Coordination, 2007 Future of Software Engineering (FOSE'07), Minneapolis, Minnesota, May 23-25, 2007.

[5] E. Hustad, Knowledge Networking in Global Organizations: The Transfer of Knowledge, *SIFMIS,* Tucson, Arizona, USA, April 22-24, 2004.

[6] N. Jastroch, T. Marlowe: Knowledge Transfer in Collaborative Knowledge Management: A Semiotic View; **Journal of Systemics, Cybernetics and Informatics**, JSCI, Vol. 8, No. 6, pp. 6-11, 2010.

[7] N. Jastroch: Advancing Adaptivity in Enterprise Collaboration*,* **Journal of Systemics, Cybernetics and Informatics**, JSCI, Vol. 7, No. 6, pp. 7-11, 2009.

[8] N. Jastroch: Adaptive Interenterprise Knowledge Management Systems. Proceedings of The 12th World Multi-Conference on Systemics, Cybernetics and Informatics: WMSCI2008, Vol. VII, IIIS Publication, Orlando/FL, 2008

[9] N. Jastroch, V. Kirova, C. S. Ku, T. J. Marlowe, M. Mohtashami: Software Engineering Must Be Collaboration-Aware, (Position Paper), Proc. of the 22nd International Conference on Software and Systems Engineering and their Applications [ICSSEA], Paris, France, December 2010.

[10] N. Jastroch, V. Kirova, C. Ku, T. Marlowe, M. Mohtashami, Adapting Business and Technical Processes for Collaborative Software Development, Proceedings of the 21st International conference on Concurrent Engineering, ICE 2011, July 2011

[11] V. Kirova, T. Marlowe : "Prendre en compte les changements dynamiques dans le développement cooperative du logiciel", **Génie Logiciel**, Decembre 2008, Numéro 87, pages 15-25.

[12] T. J. Marlowe, V. Kirova: Addressing Change in Collaborative Software Development through Agility and Automated Traceability, WMSCI 2008, 209–215, Orlando, USA, June-July 2008.

[13] T. J. Marlowe, V. Kirova: High-level Component Interfaces for Collaborative Development: A Proposal, **Journal of Systemics, Cybernetics, and Informatics***,* 7 (6), pages 1-6, 2009.

[14] T. J. Marlowe, V. Kirova, N. Jastroch, M. Mohtashami, A Classification of Collaborative Knowledge, Proceedings of the 4th International Conference on Knowledge Generation, Communication and Management: KGCM2010, Orlando FL, July 2010.

[15] M. Mohtashami: The Antecedents and Impacts of Information Processing Effectiveness in Inter-Organizational Collaborative Software Development, Ph.D. Thesis, Rutgers University, July 2006.

[16] M. Mohtashami, T. Marlowe, V. Kirova, F. Deek: Risk Management for Collaborative Software Development, **Information Systems Management**, 25 (4), 20–30, Fall 2006.

[17] M. Mohtashami, T. Marlowe, V. Kirova, F. Deek: Risk-Driven Management Contingency Policies in Collaborative Software Development, **Intl. Journal of Information Technology and Management**, to appear, 2011.

[18] M. Mohtashami, T. Marlowe, V. Kirova, F. P. Deek: A Comparison of Three Modes of Collaboration, 15th Americas Conference on Information Systems (AMCIS 2009) [CD-ROM], August 2009.

[19] M. Mohtashami, T.J. Marlowe, V. Kirova: Lost In Translation: When Outsourcing Decouples Development, 41st Annual Meeting of the Design Sciences Institute (DSI 2010) [CD-ROM], San Diego CA, November 2010.

[20] N. Schadewitz: Cross-Cultural Collaboration, http://crossculturalcollaboration.pbworks.com/FrontPage, accessed October 2010.

[21] M. J. Schniederjans, A. M. Schniederjans, D. G. Schniederjans: Outsourcing and Insourcing in an International Context, M.E. Sharpe, New York.

[22] J. Sutherland: Future of Scrum: Parallel Pipelining of Sprints in Complex Projects, AGILE 2005 Conference. Denver, July 2005.

[23] J. Whitehead: Collaboration in Software Engineering: A Roadmap, 2007 Future of Software Engineering (FOSE'07), Minneapolis Minnesota, May 23-25, 2007.

[24] J. Whitehead (ed.): Collaborative Software Engineering, Springer, 2010.