Adapting Business and Technical Processes for Collaborative Software Development

N. Jastroch¹, V. Kirova², C.S. Ku³, T.J. Marlowe⁴, M. Mohtashami⁵

¹MET Communications, Bad Homburg, Germany, norbert.jastroch@metcommunications.de
²Alcatel-Lucent, Murray Hill, NJ, USA, vassilka.kirova@alcatel-lucent.com
³William Paterson University, Wayne, NJ, USA, kuc@wpunj.edu
⁴Seton Hall University, South Orange, NJ, USA, thomas.marlowe@shu.edu
⁵Advanced Infrastructure Design, Hamilton, NJ, USA, mojgan@aidpe.com

Abstract
As software development becomes more collaborative, business and technical processes of software engineering and their management need to accommodate and support collaboration. In this paper we briefly survey key concerns, known challenges, and potential alternative solutions. We draw on lessons learned from software process models, organizational behaviour and knowledge management and their relevance to collaborative software development. We suggest elements of the solution to the problem of managing business and technical processes with software development in inter-organizational context.

Keywords
Software architecture, collaboration, agility, components, interfaces, knowledge management, software engineering

1 Introduction

Software projects are already intrinsically cooperative, requiring many software professionals to coordinate their efforts and to collaborate at many levels to produce a large software system. Integral to this effort is establishing shared mental models, achieving alignment on the system objective, and building common understanding of the shared artefacts created throughout the development process. Continuing trends for outsourcing and off-shoring, subcontracting and forming of joint ventures, as well as academic-industrial collaboration, expand the collaboration scope and raise new challenges and numerous opportunities. While legal issues and management support appear to be the most substantial areas, we discuss many other issues and contend that all aspects of software development need to be re-examined to effectively address the collaboration needs. These issues are complementary to selection and establishment of a good collaborative tool and information environment. Although a good tool and information environment is critical for successful collaboration, tools address the question of how collaborative work is to be supported, whereas here we consider what modifications of standard policies, artefacts and processes are needed to enable initial and permit optimization of future collaboration.

A key challenge in multi-organizational collaboration efforts is that responsibilities need to be assigned early in the project inception phase, which requires the scoping, decomposition of the product and the definition of the overall project structure to be established well before the application to be developed is even marginally understood. That, especially in large, complex or innovative projects, in turn requires flexibility in interactions between teams and partnering institutions. It further requires flexibility in defining the allocation of responsibilities to the components, subsystems or features they are to deliver. Given the constraints of collaboration, it is crucial to understand that the best high-level decomposition may not be the optimal structural or functional decomposition, but will also be driven by management objectives, technical expertise, resources, and other considerations in the individual organizations. We also consider
issues related to areas such as requirements and knowledge management, architecture and testing, evolution and intellectual property.

A notable current trend in both the technical and management aspects of software development is agility. Agile frameworks deal well with uncertainty and accommodate changing requirements through ongoing customer involvement, short iterations, open communications, and the application of practices such as retrospectives and steady re-factoring. They are further characterized by self-organizing cross-functional teams, application of technical practices such as test-driven and acceptance-driven development, test automation and continued integration. The projects also benefit from avoiding unnecessary specification and upfront extensive analysis. Agile methods bring the much needed flexibility, but can they address all the challenges of multi-organizational collaboration efforts?

In the balance of this paper, we first consider three motivations for modifying processes - software processes and process models, organizational behaviour, and knowledge management issues. We consider in particular one common situation - a mediating agent between customer and developer. Such agents are often employed when the software development project is part of a larger computer-based or computer-aided business system; we will refer to this mode of collaborative development as managed development. Next, we present some general concerns and some limited recommendations for adapting and optimizing software development for collaborative ventures. Finally, we briefly consider related work and present our conclusions.

2 Lessons for Collaborative Software Development

Collaborative software development is different both from single-developer software development projects and from other collaborative ventures. We present three perspectives: software process models, organizational behaviour, and knowledge management.

![Figure 1. ICSD perspectives](image)

2.1 Lessons from Software Process Models

When software development is coordinated by an agent, requirements elicitation and specification are separated from design and implementation; integration and most testing are often deferred until component milestones are achieved for all or most modules, and communication with the client is through a third party (the agent), not immediate, and sometimes infrequent. This lack of direct communication and the delay in exchanging information can significantly hinder the quality and lead to wasted effort. Further, we have found [32] that use of formal traditional development approaches [36] can actually hinder collaboration, specifically when interacting system components have different developers.

The software engineering community has come to the consensus that the traditional heavy-
weight, typically sequential (waterfall-based) process models are neither effective nor required when developing complex software intensive systems, and particularly when any of the following characteristics apply:

1. The requirements are initially imprecise or volatile, or the degree of theoretical or technical innovation and discovery required is high.
2. The application is to be long-lived and evolving. It is expected to work on multiple platforms, to interact with multiple providers for the same services, and to deal with changing requirements, both during development and after deployment.
3. The system is complex, with multiple autonomous but interacting subsystems and services which can evolve independently.
4. The knowledge of the target business domain and its work-flows, or the application domain and its characteristics such as security or distribution, require substantial tacit and implicit knowledge, or integration of knowledge from multiple sources.
5. The delivery of the system features is expected to occur in small increments
6. The components and other artefacts developed for the application are expected to be reusable with minimal effort, and are themselves valuable intellectual property.

In such cases, less rigid, more flexible development models are preferred, able to respond to refinements of or changes in requirements - compare also [28], and incremental development outward from core and high-risk functionality. Depending on the level of risk, the level of innovation, the structure and expertise of the participating teams, as well as the expected duration of the development efforts and the stability of the collaborative venture, the models can range from relatively heavy-weight (that is, artefact- and coordination-intensive, for example, classical Unified Process [13, 20]) to lighter-weight agile processes such as SCRUM [21, 22, 23, 24].

Nonetheless, the application of agile methods [3, 5, 17, 21, 26, 31] in any broad collaboration needs attention and custom selection of most suitable agile management and engineering practices [24]. Since these methods rely on rapid turnaround and pervasive client contact, prioritization, evaluation, and feedback, the collaborative model needs to be augmented with open communication channels and light-weight artefacts as a shared product backlog [37, 40].

2.2 Lessons from Organizational Behaviour

Studies of team and organizational behaviour, together with studies of collaborative software development have established several factors influencing both product and project success [2, 12, 33], either for collaborative projects in general, or specifically or more intensively for inter-organizational collaborative software development (ICSD). Here we consider in particular those factors that extend to collaborative software design in general, in the absence of careful design or selection of policies, artefacts, processes, and activities, or when relevant information is not shared between partners and/or subcontractors.

Perhaps the two most important general factors are trust and communication [1, 11, 35]. We have already seen that three-way communication is likely to both introduce delays and degrade quality. Trust also suffers. First, the lack of management contact between client and subcontractor is likely to interfere with formation of a shared management vision and continued management buy-in. Second, imprecision and delays in relayed information reduce trust on the technical side.

When there are multiple subcontractors, and when significant integration and standards [39] for the resulting components are required, additional factors specific to ICSD and impacted by the managed integration scenario include the need for:

1. Early integration, evaluation and testing to prevent serious delays and/or late discovery of faults, and software configuration management for the project as a whole, with the supporting artefacts needed for debugging and change management.
2. Coherence in the software process, either through use of compatible models or through creation of compatible views of information needed for proper integration, including management of knowledge relating to the software development process.

3. Proper levels of flexibility and agility, combined with well-specified and enforceable interfaces [18].

2.3 Lessons from Knowledge Management

Collaborative software development projects present challenges in knowledge management arising from collaboration, software development and deployment, and their interaction. These complicate ICSD in at least four significant ways (compare [4, 14, 15, 16, 30]).

1. First, unlike, say, a machine part, a software system makes significant use of knowledge not only in its design, but in its use, and in its evolution. Hence the need to take into view design time vs. run time development issues. Some of these have been discussed in [16] for the case of the engineering of inter-enterprise knowledge management systems: (1) the modelling of business and technical processes taking place at the time of design of a software system must prepare also for its run time adaptability; and (2) the use of evolving knowledge in a collaboration requires semantic non-ambiguity as a prerequisite [14]. Collaborative software development thus is challenged to generate adaptive business and technical process models first, and then provide means for semantic synchronization, which frequently will include the necessity to align existing knowledge bases of collaborating partners.

2. Second, at least some of that information will not belong to a single partner, but will require integration of knowledge from multiple partners and external sources, particularly in proper specification of interfaces and interactions. It is essential here to be careful in alignment of that knowledge to account for the different connotations of the several partners [14]. This can be done on the formal system level, using glossaries and interfaces; or through ongoing interaction of collaborating partners. Specific features analogous to the role of a mediator then apply. The relevance of such role type features has been pointed out in [15]. In collaborative software development, thorough analysis of the different roles, their specific features and their impact on collaborative practice will be a fundamental success factor.

3. Third, the design of the software, and more particularly its use, can produce large amounts of information. This information has obvious business uses, but can also be used to optimize the product, in particular to improve program robustness, efficiency, usability, and functionality. This is subject to an explicit knowledge augmentation strategy in any case where the economic benefits of an evolving knowledge base are to be fed into the business of any of the collaborators. In fact, this kind of emergent collaborative knowledge is an additional advantage of collaborative software development. In [30], the need for collaborative examination of emergent knowledge and changing perception of information has been stated as an issue to address with respect to knowledge re-evaluation and evolution.

4. Finally, dual to the second point, much of the information produced cannot be said to belong to any one of the partners, and some information may even require the talents and knowledge of multiple partners to interpret. A common knowledge exploitation strategy must therefore be agreed, to enable harnessing of these benefits. This can be done either anticipatively, or in parallel to the generation of exploitable knowledge during collaboration. The latter approach creates the need to, at least partially, continue collaboration even after the collaborative development has been completed.

These issues not only complicate knowledge management itself, but also impact risk management [33, 34], management contingency policies, and legal issues including intellectual
property and security. In fact, they also offer opportunities to derive, from collaborative engineering activities, input to product customization and innovation, or to product life cycle management, and so benefit the economically exploitable knowledge base of partners in a collaborative enterprise.

3 Activities and Work-flows in the Collaborative Context

In [19] we have considered a number of policy, artefact, and process issues that need to be resolved to foster and optimize both the eventual products of a collaborative venture and the health of the collaboration itself. Addressing these issues complements rather than competes with selection and establishment of a good collaborative tool environment. Tools address the question of how collaborative work is to be supported, whereas our concern here is focused on what needs to be done to allow the process, product and partnership to succeed.

Here we just recall the listing of these issues as they have been addressed in [19]:

![Figure 2. Issues to resolve in ICSD](image)

4 Managed Integration: Elements of the Solution

In this section, we consider four possible solutions, each of which might address part of the problem, and indicate why even if in combination they are likely to improve the situation, but unlikely to be sufficient in themselves. The first two of these modify the roles in the three-way relationship; the other two call for some software process knowledge on the part of the agent.

4.1 Direct Developer-Client Contact

Direct contact between client and developer will of course reduce or even eliminate difficulties in filtering, translating, and relaying information. One problem is that this contact threatens to bypass the domain expertise and evaluation functions of an agent, as well as the agent’s mediation in dealing with difficult clients and in handling cultural and linguistic differences. Furthermore, an inherent problem exists with multiple subcontractors, since one or two will have to become responsible for client contact, and therefore have to deal with the other subcontractors, which may be out of their expertise. This approach has to be used with care, with limitations on the mode and content of the exchange. However, these risks are ameliorated to the extent that the agent shares development or integration responsibility (see Section 4.4 below).

4.2 Flexible Interfaces

Modified component interfaces, such as those described in [18, 29], including artefact flow
management, can be used to limit the scope of problems arising from changes in requirements or discovered during detail design, coding or testing. Full use of these interfaces will also partially automate integrated software configuration management, and perhaps assist in creating a software component repository available to all collaborators, which can aid in ensuring a standard architecture. However, there are several costs. First, additional knowledge of software development and perhaps additional domain knowledge will be required at either the client or the agent. Second, developers may have to modify software development processes to be able to take advantage of this approach. Third, the agent and the developers will be required to modify IT processes and artefacts. Finally, there are additional costs in defining both the components and the framework for interaction.

4.3 Early Prototype or Integration

Many of the problems associated with imprecisely and partially specified requirements can be avoided if in-house development of an early prototype is supported. A prototype will allow for early feedback and richer interactions with the customers addressing some of the software process issues discussed earlier, but it might not be consistent with the final integrated product. On the other hand, early and ongoing integration may require upfront investment and resources, but will allow for early problem detection, making those problems more comprehensible. It is also likely to allow to flag unqualified, inadequate or insincere efforts from new subcontractors and to request input and clarifications from the customers on an ongoing basis.

4.4 Management Implications

These considerations reinforce the importance of mediating management at the boundaries of allied organizations (see Figure 3 [34]) to facilitate transfer of knowledge, and also to monitor and coordinate scheduling, promote streamlining and championship, and provide conflict resolution and risk management.

In the managed development mode, this responsibility falls almost entirely upon the mediating management. The mediating management therefore needs to have a good understanding of both domain and technical requirements and the surrounding software issues, as well as a great deal of management knowledge. In particular, the client representative should have in-depth knowledge of establishment and use of implicit and explicit process flow, shared mental models, and work familiarity [7]. Any shortcomings in these areas are likely to complicate, delay, and possibly prevent completion of the software development project.

5 Related Work and Conclusions

Most of the existing literature on collaboration considers intra-organizational collaboration, or
focuses on selected aspects of inter-organizational interaction. This paper in contrast specifically addresses inter-organizational collaboration from a broad, multi-faceted and systemic point of view. Incorporating agile practices in intra- or inter-organizational collaboration is discussed in [2, 8, 9, 25, 27]. Herbsleb [10] and Whitehead [42] address the problems of collaboration, but largely within a single organization, and primarily limited to tool support and software configuration management - see also [41], including expansion of the set of desirable artefacts. Erickson [6] and Schadewitz [38] provide patterns for component interfaces and interactions. But, other than in our previous work, there seems to be little explicit focus on, for example, collaboration-aware metrics, changes in testing, knowledge management for collaborative software development, or risk management per se.

In conclusion, we have addressed the implications of inter-organizational collaboration for a number of development aspects - including both core software engineering and umbrella activities. Inter-organizational development and collaboration for large, complex or innovative software products calls for new approaches to accommodate both organizational differences and flexibility in development. As we discuss, all aspects of software development - technical, process, and management - are affected. This paper outlines pressing issues and presents initial or partial solutions in several areas.

References