Using Iterative and Incremental Processes in Global Software Development

Maria Paasivaara and Casper Lassenius
Helsinki University of Technology
Software Business and Engineering Institute
POB 9210, FIN-02015 HUT, FINLAND
Maria.Paasivaara@hut.fi, Casper.Lassenius@hut.fi

Abstract

Iterative and incremental development seems to be a viable approach providing several benefits in inter-organizational distributed software development. This paper presents initial results from an interview study on the usage of iterative and incremental development in inter-organizational distributed software development projects. We describe identified practices, such as delivery synchronization, design and code reviews, communication emphasis, feature-based development, behavioral patterns, and frequent deliveries. We also present the benefits that the use of these practices brought, such as transparency of progress, increased developer motivation due to rapid feedback, flexibility regarding changes, the possibility to involve subcontractors early, ensuring joint understanding of requirements, and the avoidance of "big bang" integration. It seems that the advantages of using the practices overweigh the extra communication and coordination cost they incur.

1. Introduction

Global inter-organizational software development, including outsourcing, subcontracting and partnerships, is becoming increasingly common. Projects developing genuinely novel products are often faced with uncertainty regarding, e.g., requirements and implementation technologies. However, subcontractors or partners often need to be involved long before these uncertainties can be resolved. In such projects, the parties cannot receive clear requirement specifications at the beginning. Instead, close cooperation and communication between the parties is required during the whole project, as the project both builds a product and tries to understand what to build at the same time. In these kinds of projects, problems often arise, since practices and processes needed for collaborating across distances and organizations are neither well understood in theory, nor typically established in practice [9].

For software development facing uncertainties and unpredictable changes, literature suggest the use of iterative and incremental development (IID) as a process model, since it enables fast reaction to changes [6]. IID means that the system is grown via iterations, incrementally adding new features [6]. Global software development literature contains some reports of good experiences of using IID also in distributed settings (e.g. [1, 2]). However, these studies do not report in detail how IID should be implemented and used successfully in distributed projects nor what benefits and drawbacks its use brings.

In our interview study concentrating on collaboration practices in globally distributed projects, we noticed that IID was used in many of the projects studied and that these projects had gained several advantages from its usage [9]. It seems that IID suits distributed development extremely well and helps reduce problems caused by distribution. However, IID also requires close collaboration and communication, which can be hard to achieve in distributed development.

In this paper we report experiences of using IID in five globally distributed inter-organizational software development projects. We present some interesting findings of how companies are using IID in their distributed projects and what kind of benefits they have gained. Since our larger study concentrated on all collaboration practices used in these projects, we could not yet go very deep into IID related practices. The purpose of this paper is therefore to give an overview of our findings related to the usage of IID in distributed development, as well as to motivate further research into its use, benefits and drawbacks.

The rest of this paper is structured in the following way: The next section briefly presents related literature. After that we describe the research methodology and introduce the case companies and projects studied. In the results section we present the experiences we collected from our case projects. Finally, we present a short discussion of the results and their managerial implications, as well as give ideas for future work.

2. Related Work

Global software development literature lists many challenges related to distributed development, e.g., interdependencies among work items that are distributed, difficulties of coordination, difficulties of dividing the work into modules that could be assigned to different locations, conflicting implicit assumptions that are not noticed as fast as in collocated work, and communication challenges [8]. Literature suggests dividing the work into separate modules that can then be distributed to different sites to be developed [4]. These modules should be so independent that communication between sites can be minimized [4]. The authors emphasize that it is possible to split only well-understood products where architecture and plans are likely to be stable. However, in a development environment with a lot of uncertainties, dividing the software into modules and specifying the modules in detail up front is often impossible. Moreover, first specifying and dividing work and subsequently integrating all in "a big bang" is challenging, since integration can cause huge unexpected problems. As a solution Battin et al. [1] suggest an incremental integration plan, which is based on clusters and shared incremental milestones to avoid "big bang" integration. This strategy was used successfully at Motorola. Ebert and De Neve [2] provide similar experiences on the usage of incremental development at Alcatel, where they developed each increment within one dedicated team and based their progress tracking on successfully integrated and tested customer requirements. The authors report that a stable build proved to be one of the key success factors and that the globally applied continuous build improved the project's cycle time. Neither Alcatel nor Motorola report their integration intervals, but it seems that they did not use very frequent regular build cycles, such as daily or weekly builds. However, even very frequent builds are possible in distributed development. Karlsson et al. [5] report using daily builds and feature-based development successfully in distributed projects at Ericsson.

IID is a core practice in agile methodologies for collocated projects [6, 7], but its use in distributed development has not yet received much attention. Fowler [3] and Simons [11] recently reported their experiences on using agile methods in offshore software development projects. According to Simons [11], an iterative model seems to work well in distributed projects and can eliminate some of the problems that distribution brings. Continuous integration and build verification tests solve integration problems in small steps and avoid large integration problems at the end of the project. Moreover, IID provides increased visibility into project status, which makes it easier for project managers and customers to follow project progress [11]. Fowler [3] discusses the suitable iteration lengths for offshore projects and concludes that iterations cannot be shorter than two weeks, because of the communication overheads of distributed development, and two to three month iterations can already be too long.

This short literature review shows that IID seems to be a viable process model for distributed software development projects, providing several advantages. However, the reported experiences of its use in distributed environment are still quite limited. We believe that collecting more real-life experiences of the usage of IID and the gained advantages would be helpful to managers designing their distributed projects. In this paper we report our initial research results, discuss the benefits of IID in distributed inter-organizational development, and outline ideas for future research topics.

2.1. Research Methodology

The research presented in this paper follows the case-study approach [12] and is a part of a larger multiple case study [9]. The aim of the larger study was to collect successful collaboration practices from inter-organizational software development projects. We used purposeful sampling [10] and selected ten projects from eight companies that we knew used software subcontractors and that we expected to be experienced in inter-organizational software development. We selected projects that demanded constant collaboration and lots of communication between the parties, e.g., due to a high degree of uncertainty, dependencies and changing requirements.

One of the successful practices we found in the larger study was the use of iterative and incremental development. From the ten projects we studied, five used an IID model. In this paper we report experiences collected from those five projects. In these five projects we performed 29 semi-structured interviews, each lasting 2–3 hours.

After our first interview round and data analysis we noticed that IID was a central theme in these five projects. We also noticed that project A was the most interesting project in this sample regarding IID. Therefore we chose to do one extra interview with a manager from that project concentrating only on experiences of IID. We had interviewed this person also during our first interview round and wanted to ask more detailed questions on IID. Basic information about the projects and the number of interviewees is shown in Table 1. The next paragraphs provide short descriptions of the studied projects.

Project A was a new product development project with lots of uncertainty concerning requirements and technology. The German office of a Finnish customer company did this project with the help of two new subcontractors, one from Germany and one from Ireland.

Project B was from the same Finnish customer company as project A. This project also contained a lot of uncertainty

Table 1. Case project interviews

Case projects		A	B	C	D	E
# of interviews	Partnership manager	1	2	1	-	1
	Process developer	1	1	-	-	1
	Project manager	1	2	3	2	3
	Team member	-	-	2	-	1
	Sub-contractors	-	1	5	-	1
	All	4	5	11	2	7
Industry		Tele-com	Tele-com	Fi-nance	Inter-net SW	Be-spoke SW
Distribution (# of sites)		Eu-rope (3)	Eu-rope (4) & North Amer-ica (1)	Eu-rope (3)	Eu-rope (1) & Asia (2)	Eu-rope (6)

regarding requirements and technology. Projects A and B were both subprojects of larger product programs. Project B used a Finnish subcontractor with sites in Finland and Hungary. The customer company had sites involved both in Finland and in the US.

Project C was a a large new product development project done by a Finnish company. Additional resources were assigned from its newly acquired subsidiaries in Denmark and Switzerland.

Project D was a development project for a customer specific system carried out by a small Finnish company, which had its sales and project management in Finland. All development work was performed in a partly owned subsidiary in India.

Project E developed a well-defined customer specific system and was distributed to six sites. In addition to the Finnish customer company's three own sites, also a subsidiary in Estonia and a subcontractor with two sites in Finland were involved.

3. Results

In this section we first introduce our findings on how companies were using IID in distributed settings. The companies used practices such as synchronization of deliveries, design and code reviews, emphasis on communication, feature-based development, behavioral patterns, and frequent deliveries. Then, we present some of the benefits of using IID, such as transparency of progress, increased developer motivation due to instant feedback, flexibility to do changes and start early with subcontractors, ensuring understanding of requirements, and the avoidance of big bang integration. To summarize, our results seem to indicate that IID is a suitable approach for distributed projects and that more detailed studies on the methods and practices of using it are needed.

3.1. Identified Practices

Delivery synchronization In an inter-organizationally distributed project different partners might have different delivery and integration cycles, but our case studies show that synchronizing the delivery and integration cycles between participants is beneficial.

In project A the original plan was to use the waterfall model, but after some quality and schedule problems the customer company and the Irish subcontractor started to use an iterative development model with weekly builds. However, getting used to this new weekly rhythm was not easy. Especially coordinating the work between different teams and specifying the work well enough required learning. The customer tested each build once a week for one day and after that everybody got this tested build as a new baseline. During the early phases of iterative development the teams learned that it was better to develop only small additions at a time to avoid problems. The German subcontractor delivered in longer intervals, only once in 1-3 months. This caused additional work in the integration phase, because the amount of new code was large and not always compatible with the baseline. Finding bugs from 1–3 months worth of work was not easy. In this project the baseline was available to everyone through a common repository or its replica. This made it possible for all partners to test their new code against the baseline before integration.

Project B faced problems with delivery cycles of different length, since two sites used one-week iterations, and one site had a two-month cycle. This led to problems for the subcontrator, since it ended up waiting up to two months for fixes from the customer site using long iterations.

Our interviewees from projects A and B emphasized that when using IID it is important that all participants synchronize the iteration cycles, i.e., use the same length for iteration cycles and deliveries. If this is not done, problems will occur.

The iteration and delivery cycles used varied between projects and also between project phases, e.g., in the beginning they could be longer and in later intensive phases they were shorter. In project B, early phases used three month increments, and the late phases weekly deliveries.

Design and code reviews Design and code reviews seemed to be useful in distributed projects with distant sites or subcontractors. These reviews are early checks that the distributed teams have understood the requirements correctly and are doing what they are supposed to do. In later

stages, the deliveries of code fulfill this need. The distributed sites also felt that these reviews were very useful since they got immediate feedback on their work.

In project C iterations were used only with the company's Swiss subsidiary. Their work consisted of three months work with two planned iterations. This project was the first collaboration effort after the Swiss subsidiary had been bought, therefore starting the project required similar efforts as with subcontractors. Before the coding became intensive the customer's Finnish contact person visited the Swiss team twice, first having a design review and then a code review. After that the implementation could safely start and everybody knew that the work was on the right track.

Also in project E code reviews were used in the early phases of the project with the subcontractor and the foreign subsidiary. These reviews were very much appreciated since developers got immediate feedback on their work.

Emphasis on communication Communication requirements in distributed projects using IID and therefore collaborating closely are very high. Especially the projects that had weekly integration cycles, A and B, found communication as a very important prerequisite to be able to work that fast.

Project A had weekly integration meetings, where integration related problems were discussed. These meetings made it possible to learn from mistakes already in the early phases of development. Project progress monitoring also took place during these meetings: only tasks that had passed the tests and been integrated into the build were regarded as ready. Subcontractors could not participate in these meetings, because of security issues concerning this totally new product, but the customer had project managers that represented each of the subcontractors in the meetings and delivered information to the subcontractors. In this project only the Irish subcontractor participated in the weekly cycle. Frequent communication with this subcontractor was ensured by having their staff sitting at the customer's premises. Ad-hoc communication and meetings were encouraged in this project. Also the "behavioral patterns" used in project A, and described later on, are closely related to emphasizing fast communication and getting answers quickly.

Project B had a normal weekly face-to-face meeting in all its teams. The following day project managers both from the customer company and the subcontractor had a weekly teleconference. The subcontractor's team leaders could also participate in this meeting if they deemed it necessary; otherwise they could read the meeting memos.

Both projects A and B had higher-level monthly meetings. In project A they were called R&D meetings. In Project B they were project steering group meetings, which were organized every time at different project sites to enable both managers and developers from the different sites to meet face-to-face.

Project C, having only two iterations, also found frequent communication to be an important factor for the project's success. The Finnish customer company had one person responsible for all communication with its Swiss subsidiary. This person felt that his task was to answer all questions as soon as possible. These fast answers were very much appreciated by the Swiss developers. Moreover, this contact person had three collocated stays with the team in Switzerland, each lasting about one week. This, of course, facilitated communication and built trust, which was also regarded important.

Project D delivered a customer specific system using IID. The main contractor was a Finnish company that used its partly owned Indian subcontractor as a development resource. The Finnish office negotiated the requirements with the customer, made a requirements specification document and delivered it to India. The subcontractor's project manager commented on the requirements by email and asked detailed questions. The Finnish project manager answered the questions by email and discussed difficult issues through chat. The aim was not to create a perfect specification, since the project's customer could not provide that. Instead, the project was specified to such a level of detail that the Indian subcontractor could develop an initial version of the system. After the delivery of this initial version the Finnish main contractor commented on it. And then, after some improvements, also the Finnish customer commented on the system. The project had several of these comment-improvement rounds. During the whole development, the Indian developers were encouraged to ask questions through chat from the Finnish main contractor's project manager. The customer was also able to monitor project progress by reviewing the code that the Indian developers checked in to a repository located in Finland several times a day. This well functioning communication process was used in all projects between this main contractor and its Indian subcontractors.

Feature-based development Project A had clearly separate sub-areas that could be given to each of the subcontractors. Because builds were weekly and the customer wanted to do functionality testing, the work had to be coordinated quite tightly so that all code affecting certain functionality would be ready at the same time. This feature-based development meant that small increments done in different modules had to be evolving in good synchronicity, in order to enable proper testing and to avoid difficult merging of code later on. In Project A, the customer's project manager made a monthly plan of the tasks to be performed, and the subcontractors' project managers made weekly plans of their internal tasks.

Behavioral patterns Project A had noticed that using an IID model in distributed development did not automatically bring all necessary practices needed for successful cooperation. This project developed additional practices they called "behavioral patterns" to complement the development model. These practices cannot easily be described in process descriptions but are very essential to IID according to our interviewee from project A. He ensured that in cooperation with subcontractors and partners these practices have been equally valid and important as in internal development. According to him, project A had 16 behavioral patterns, that were developed during the project concerning, e.g., management, personal behavior, and the use of tools and work processes. Our interviewee presented three examples of the practices that, based on his experience, were important for the success of their very short-cycled IID process.

A practice called "immediate escalation of issues", means that problems have to be brought up right away. The "project manager always available" practice is closely related to the previous one, meaning that when a developer gets stuck he can immediately ask for help from the project manager who has to be available. "Immediate decisions" means that decisions have to be made fast and not left to later meetings, so that work can continue. This last practice was felt to be more difficult to use across distances when people making decisions might never have met face-to-face and this can easily lengthen decision-making.

Frequent deliveries Project E used frequent deliveries when designing and implementing a large customer specific system. The requirements were quite stable and well-known. The customer company divided the work into small tasks and specified, e.g., all windows and services in detail. These well-specified tasks were then given to subcontractors and internal sites for implementation. Specification work and coding took place at the same time. Both a subcontractor company and an own subsidiary received tasks for 2–3 weeks at the time. When the tasks were done, a delivery was made, and new tasks were assigned. The problem with this way of working was that these delivered services and windows had dependencies to other services or windows, and the customer company could test them only after all related functionality was ready. Therefore, getting test results could sometimes take as long as half a year after code delivery. Clearly, this project would have benefited from better synchronization of deliveries.

3.2. Benefits gained

Transparency of progress Frequent delivery cycles and integration brought transparency of work progress to all partners. When both the customer and the subcontractor used IID, the subcontractor regularly delivered functioning code during the development phase, e.g., monthly or even weekly. Our interviewees told that when deliveries were integrated and tested right away this gave a very good picture of how the project was progressing. They had noted that frequent deliveries made it easier for the customer to monitor the real progress of the subcontractor's work.

Instant feedback Integration and testing reports gave distributed developers instant feedback on their work, which they felt was very motivating. Moreover, when the customer saw that the subcontractor was doing a good work, the customer's personnel started to trust and respect the subcontractor and its developers' know-how, which made further collaboration easier.

Flexibility From the customer's point of view IID brings additional flexibility, when the customer can do changes also during the development phase without time consuming negotiations with subcontractors. Of course, a suitable type of contracting has to be chosen. IID also enables the customer company to take subcontractors into the project already in the early phases of development, when requirements cannot yet be specified in detail. With this kind of development process it is no longer necessary to specify all requirements before subcontractors are involved; instead, since requirements are allowed to change during the project, work can start despite technological or goal-related uncertainties. However, this requires all parties to have "an experimental mindset" and fast and open communication with each others.

Ensuring joint understanding of requirements In IID frequent integration and testing ensured that the subcontractor had understood the requirements correctly. This is a typical uncertainty in distributed development, especially when companies have not worked together before and have different cultures. Frequent integration and testing gives fast feedback and any misinterpretations become visible early. Thus, possible misunderstandings have less damaging consequences. Moreover, learning from mistakes is fast and happens early, preventing problems from accumulating and creating situations that are harder to resolve.

Avoiding "big bang" integration IID,as used in our case companies, prevented different sites and partners from doing too long periods of independent development, which could have led to modules that would be hard or impossible to integrate, i.e., they avoided possible problems that would come from "a big bang integration".

4. Discussion and Conclusions

Frequent communication is a central prerequisite for successful implementation of IID in a distributed environment. This way of development requires much more communication between parties during the whole collaboration compared to development where work products can be clearly separated into independently developed modules. Especially uncertainties in the form of changing requirements demand communication and problem solving. In uncertain environments short iteration cycles are needed to reveal problems as early as possible. The shorter the cycle the more communication is needed to coordinate the work and to quickly solve the problems during development. When the cycles are longer the communication need is more concentrated to the integration phase. This communication overhead is clearly an issue that requires careful planning when implementing IID in a distributed environment.

Defining an iteration length that is suitable for distributed development is an interesting topic. In our study both projects A and B, used one week integration cycles successfully also with subcontractors. Fowler [3] reported that iteration cycles should not be shorter than two weeks in offshore projects due to communication overhead. One explanation to this difference could be that in our study all parties that participated in weekly iterations were from Europe and therefore the time-zone difference was quite minimal. Moreover, the subcontractor in project A had on-site personnel at the customer company, which facilitated communication. Fowler, however, worked with projects distributed between India and Europe or North America, where time differences are noteworthy.

4.1. Limitations

Our initial results presented in this paper are based only on a few case projects, which limits our possibility to draw far reaching conclusions. Moreover, when doing our first interview round we did not concentrate on studying IID, but asked about many other practices too. This means that we could not collect as deep a knowledge on IID and related practices that a more focused study could have provided. The selection of our case projects did not concentrate on finding interesting cases just from the point of view of IID.

4.2. Managerial Implications

We think that our results have several implications for managers working with distributed development. First of all, it seems that short increments are suitable for distributed development, especially when the project faces high degrees of uncertainty. However, when using increments, it is important that all partners use the same iteration length.

To enable reasonable testing of functionality, feature-based development using tight control can be used. Also the use of design and code reviews in the beginning especially with new subcontractors helps to ensure that they have understood the coding standards and requirements correctly. The feedback also motivates them. Finally, frequent communication and problem solving is essential in distributed IID. Efficient communication requires both planned communication practices and assigned resources.

4.3. Future Research

In the future we plan to study additional case projects using IID in distributed environments. We think that it is important to get deeper insights on how these challenging projects really work, what kind of practices are used, what the major problems are, for what kinds of projects this model of working is suitable. Another interesting future research topic is tool support for this kind of projects.

References

[1] R. Battin, R. Crocker, J. Kreidler, and K. Subramanian. Leveraging resources in global software development. *IEEE Software*, pages 70–77, March/April 2001.

[2] C. Ebert and P. De Neve. Surviving global software development. *IEEE Software*, 18(2):62–69, 2001.

[3] M. Fowler. Using agile software proces with offshore development. *http://www.martinfowler.com/articles /agileOffshore.html*, 7.1. 2004.

[4] J. Herbsleb and R. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 16(5):63–70, 1999.

[5] E. Karlsson, L. Andersson, and P. Leion. Daily build and feature development in large distributed projects. In *ICSE-2000*, pages 649–658. IEEE Computer Society Press, 2000.

[6] C. Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, Reading, MA, 2003.

[7] C. Larman and V. Basili. Iterative and incremental development: A brief history. *IEEE Computer*, pages 47–56, June 2003.

[8] A. Mockus and J. Herbsleb. Challenges of global software development. In *7th International Software Metrics Symposium*, pages 182–184. IEEE Computer Society, 2001.

[9] M. Paasivaara. Communication needs, practices and supporting structures in global inter-organizational software development projects. In *ICSE International Workshop on Global Software Development*, Portland, Oregon, 2003. IEEE Computer Society.

[10] M. Q. Patton. *Qualitative evaluation and research methods*. Sage Publications, Newbury Park, Calif., 2nd edition, 1990.

[11] M. Simons. Internationally agile. *InformIT*, 15.3. 2002.

[12] R. K. Yin. *Case Study Research: Design and Methods*. SAGE Publications, Thousand Oaks, CA, USA, 2nd edition, 1994.