Collaborative Team Management in Agile and Distributed Development Environments

NohSam Park and JongHyun Jang
IT Convergence Technology Research Laboratory
ETRI
Daejeon, Korea
{siru23, jangjh}@etri.re.kr

***Abstract*—The inherent nature of software engineering is collaboration. Recently software engineering practices have seen many agile methods, and distributed collaboration in geographically distant environment. In this paper, we propose the methods to manage the collaborative team for this changing environment. Collaborative team management skills in agile requires the communication skills and procedures in terms of social activities in agile process. In a distributed software project, human factors are emphasized for facilitating collaboration. The importance of risk management strategy is highlighted to address the circumstantial limitations of both environments. This paper presents the basic skills for an agile and distributed project, and reports on our experience of adapting for the real Studio project settings with the concrete methods.**

Keywords-Collaborative Team; Team Management; Agile; Distributed software development; Software Engineering.

I. INTRODUCTION

Software engineering is a result of team activity. Collaboration in software engineering has greatly increased thanks to widespread use of the Internet and many kinds of project management tools. Rapid development using agile methods also enabled various team organization and project management by emphasizing the communication process with customers [1].

Just like in many open source projects, distributed team formation may make communication more complicated because of time difference, culture, and language barriers. The wide range of engineers on the team may have different motivations and needs. These characteristics in global and diverse team management facilitate collaboration by offering technical tools and adaptive software processes. Teaming process research shows the importance of establishing and managing software teams and emphasizes the difficulties of implementing it [2]. Collaboration in software engineering refers to managing the entire lifecycle of the project, and it is the most important factor to accomplish high quality product, and efficient software engineering practices. Collaboration is complicated and hard to achieve because of the increased interdependencies between the project teams.

Agile software development has become popular since the early of 2000s, and involves collaboration and interactions naturally, resulting in creating working software [3]. The structure and organization of agile teams proves the people-focused approaches when it comes to collaboration.

The need for coordination in software project comes because tasks and artifacts between team members are tightly connected to each other, so researchers created a variety of tools and approaches to improve team coordination. In addition, some evaluation types and frameworks such as DESMET [4] for coordinating software engineering tools have been proposed [5].

Much work has been done in collaborative software engineering, but the collaborative practices are not routine and generalized. In a research field there are three main topics: theoretical understanding of collaborative software engineering, designing assessment methods for specific situations, and implementing tool support [1]. As should be clear from the practices and research work, collaboration is without doubt the core of software engineering. From the point of collaboration, it is required to develop the methods how to manage collaborative team in the current software engineering situations. As Austin and Devin described in their book [6], successfully managing knowledge workers – software team members – call for collaboration without detailed or coercive direction, keeping in mind that we cannot supervise talented employees in any conventional sense; we must lead them with passionate support and faith in their work.

This paper is organized as follows. In Section 2, some skills are proposed for enhancing collaboration in agile process and distributed development environment. Section 3 presents the case study of MSE Studio project at Carnegie Mellon University (CMU), and Section 4 concludes the paper.

II. COLLABORATIVE TEAM MANAGEMENT IN DIFFERENT ENVIRONMENT

Collaboration in software engineering has evolved through diverse processes, methodologies, and development environments. In this section, the ways of achieving the collaborative team management in the agile process and the distant development environment are discussed.

A. Agile Process Development Team

Customer collaboration and social activities get much emphasis in agile. Nevertheless, collaboration does not come naturally just by setting the agile team up. The team management skills are important in order to improve collaboration and coordination, especially between the customer and software developers in the agile process.

Identify social skills for the agile process

The agile process has key practices such as small release, simple design, refactoring, and iteration. They also put an emphasis on communication with customers and reflection on development iterations. For example, pair programming in Extreme Programming (XP) [7] encompasses the whole communication not just involving two programmers in the same room. They discuss the problem, understand the task, negotiate their opinions and share the work.

Agile process practitioners need to socialize with coworkers and customers. Most of them are familiar with communicating using social networking and instant messaging. But, socializing also requires us to keep in contact with people in the physical situation. It involves respect for the difference, understanding people's situation, and sound critique towards participants. Socializing might cause conflicts among team members whether we apply it online or offline.

The team should identify diverse social skills from many different perspectives. From the technical view, the team encourages technical discussion and research. The team can have technical workshops or open-lab for intriguing the intellectual motivation. The working condition should easily accommodate the collaboration between people. Just like XP's pair programming requires the reconfiguration of desks, the working environment should be open and shared to increase collaboration.

Establish reporting channels between stakeholders

Co-located setting of agile processes does not require formal reporting procedures to keep managers and customers up-to-date with progress. Those procedures hinder the project from moving fast, which violates the agile property. The agile principle of "barely sufficient [8]," can be applied to reporting as well. The reporting concentrates on key features developed or requirements satisfied, removing any unnecessary information. But it should be able to hold the minimum value for the project.

Agile teams need to establish the reporting channel when they show the project progress information to the customer. Many agile teams still do in a light way such as spreadsheet, sticky notes on the wall or whiteboard. The intention was not try to impose additional burden or cost on the agile practices, but it is another option for the team that wants to use agile continuously.

Many tools can provide appropriate level of information for both managers and the customer. It would be the alternative for the formal reporting procedure between stakeholders in the agile methods. Plus, agile software tools provide reflections functionality when teams finish iteration for both developers and the customer. For example, by offering the burn-down chart, it shows the simple trend and increases understanding of the project progress.

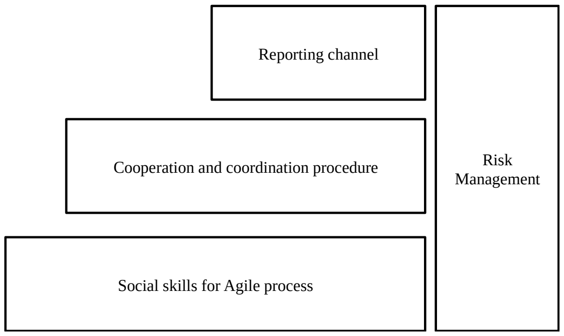

Reporting channel
Cooperation and coordination procedure
Social skills for Agile process
Risk Management

Figure 1. Collaboration skills for the agile process.

Establish a risk management strategy

The elements of the risk management paradigm are the following: *identify, analyze, plan, track, control, and communicate* risks [9]. Agile risk management follows the same activities like the traditional software projects. The iterative nature allows us to tackle high risk sooner than later. The risk management process is repeated every iteration, and remaining risks are re-assessed. Teams prioritize risks and take proactive risk management strategy for the top priority risks.

The pitfall of risk management in agile processes is that the team tends to dismiss the risks with low priorities when they assess the risks. People are likely to identify new risks for the project and focus on the high priority risks. In order to prevent the tem from overlooking those risks, the *risk overhaul* is suggested on every milestone of the project. Risk overhaul implies that existing risks are initialized and teams inspect risk management process from the scratch. From the risk identification to risk planning, teams go through every step involving the entire stakeholders. Teams can start with the remaining risks, and each risk is inspected thoroughly and reassessed.

In the risk overhaul, the outside member of the team can join with a fresh eye. In other words, every remaining risk should be treated and evaluated like newly identified risks. It could be burdensome and costly to do quite often, so it would be viable to perform it on a major milestone basis.

In Figure 1, collaboration skills for the agile process are described. Based on firm social skills, cooperation and coordination procedure should be established. On top of that, reporting channels enable the stakeholders to communicate effectively. Throughout these procedures, a proactive risk management should be implemented.

B. *Distributed Development Team*

The goal of distributed team building is to build a high performance team. Global Teaming goals are suggested in [10], each of which has specific practices and sub-practices when implementing a global software engineering (GSE) strategy. It has two specific goals: *Define Global Project Management, and Define Management between Locations*.

In distributed software development, diverse factors should be taken into account like distance, language, culture, etc. from the team setup. Especially, human factors are important for motivating participants and letting them take the initiative. The following are some suggestions for collaborative team management for distributed development teams.

Identify common goals, objectives as fast as possible

Distributed development settings require each team member to have consensus for the goals, objectives in the early phase of the project. But, the team members in different location have relatively fewer ways to get feedback and information for the project. They usually resort to online communications such as email, web-based tools, and social networking. Face-to-face interface like videoconferencing is possible, but still limited, especially when the team is globally distributed.

Distributed development teams should put much effort in getting all the stakeholders on the same page. The small problem in the early phase will snowball and end up bringing serious implications for the project. It is preferable to hold not only the kick-off meeting but also several workshops. Even though team members should be located in distance, it would be much better to get together.

Teams only work and collaborate when they share the same idea and goals. Though many technologies support meetings via audio or video, not all team members are comfortable because of diverse factors such as language barrier, time difference, etc. Just having a meeting does not guarantee to keep them agreed upon the issues. Follow-up activities should be implemented and the team should clarify the problem when they have issues during the meeting.

Define the explicit roles and responsibilities

Distant team should be given explicit roles and responsibilities for their team. Without them, the project manager will receive dozens of questions from distant development team members because they want to check what their missions and tasks are. The objective is to distribute work and motivate them to take the leadership of their own.

No one in the distant team would want to put his/her head up and lead without explicit roles and responsibilities. Make them take the initiative of the project, and make them feel they are the part of the team. When they can see what should be done throughout the project, they will make plan, accomplish tasks, and communicate as a whole team. The project manager should be able to inspire the distant team by setting the boundary of the central and distant team.

Partitioning and allocating tasks across the distant team is a key concept of the distant development. It is related to the team's capability to manage and develop features of the project. The project management should assess the distant team and local team's abilities objectively and modularize functional units.

Give autonomy and accountability

Some recommendations called "coherent and co-located teams of fully allocated engineers" were made for global software development projects [11]. They say that engineers should not be distracted by other tasks working on the same processes, methodologies, and terminology. The success of the distant development team comes from the innovation of the team members given autonomy.

The distant team can manage itself not by the central team's micromanagement. The distant team may have its own rules and management styles, thus it can make self-organized team. Then the central team gives it the necessary information, tools, and other resources in order to let it work. Product management would empower the distant team with the privilege and remove impediments in its way that may harm the progress of the project. All those things are related to promoting team performance in the project. The team as a whole can progress in its own roles and contribute to the project success.

The team needs to find the golden mean between autonomy and accountability. Autonomy should be allowed within the roles and responsibilities given by the central team. Autonomy and privileges should be only allowed in terms of the common goal: the success of the project. Autonomy naturally brings accountability for the team's result. Individuals in different locations work for the team and project's success, and each individual is responsible for their result. Use of different process can be done only when they meet the whole team's schedule, deliverables, and cost. The management should monitor and track team's progress and take actions to address the issues when autonomy gets on track of the project.

Relate the risks and problems

Distributed development projects bring additional high risk exposure as many risk factors exist such as culture-related and geographical-related risks. Bass et al. presented a coordination risk analysis method for multi-site projects in [12]. The team leader can start with this risk management strategy for the distant development project.

The team leader should relate the risks to the actual problems from these risks. It is the best to avoid risks or prevent them from becoming problems. But, some risks evade and become problems. In the risk management strategy, prioritizing and mitigating risks are highlighted, but not much attention is paid to the correlation between risks and problems when risks become problems.

Software engineers tend to either fix the problem or controlling the risk. We need to analyze the correlation between them, so that we can achieve more effective risk management. First, we analyze risk monitor, track and control activities. Then we look into what triggered the risk for becoming the problem, and what the problem's impact is. Investigating the reason and result of the problem helps us reflect on the risk management. That reflection keeps the risk with similar conditions from happening again. The risk/problem analysis process incorporates collaboration among physically distant team members.

III. Case Study

In this section, we will give an example of CMU MSE (Master of Software Engineering) Studio project, and discuss issues when team management skill suggestions are applied to the real collaborative team setting.

A. *The Studio project and team setup*

MSE is a 16-month/4 semester intensive program for software engineers. The program can be done in the form of full or part-time via distance education as well. The entire program emphasizes application of course material in a hands-on experience with real, paying clients who expect actual deliverables [13]. CMU has been incorporating the core academics of software engineering into the MSE Studio.

The Studio project has three stakeholders: the team, mentors, and the client. The team is structured as a small with three to five students from diverse culture and backgrounds. Students are expected to overcome technical challenges, and meet their client's requests through the Studio project. Mentors are assigned to each team, and they conduct, advise and guide the project. Student-mentor meetings are held weekly in an interactive style of asking the student, encouraging reflections. The client requests the development of output by giving requirements and information, providing feedback, and evaluating the deliverables from the team.

Our team was composed of five team members, two mentors, and the client. Each team member is from different country. They speak different languages, and it means the team had various factors to consider such as language and culture. Work experience was also various from less than 1 year to more than 10 years. The client of the studio project came from the area of the retail store, which has many branches worldwide. The goal of the project was to improve the customer's shopping experience such as shortening the checkout time in the local store. In order to achieve the goal, the customer required us to develop a mobile application on the Android platform.

The team adopted OpenUp [14], which is one of the agile processes, as the development process. OpenUp has 4 phases of development lifecycle: *Inception, Elaboration, Construction, and Transition*. Though the team used the agile process, the client did not co-locate in the same place with the team. In the inception phase in OpenUp, the team was supposed to refine requirements and elicit specific features for the project.

The team had to take also another thing into consideration: one of the team members had to return to the home country and continue the academics in the transition phase.

B. *Project Development and reflection about collaboration*

1) *Inception*

In the inception phase of OpenUP, the team is supposed to establish the scope of project and do the requirements analysis. The team, however, did not get much response from the customer. The client stopped communicating once in a while and the team did not take the initiative meanwhile. We should have tried to fix the problem of miscommunication and come up with our own solutions despite of the client's absence. Basically we just waited the response from the client and we did not put much effort on the Studio project, which made the agile method ineffective. In addition, we did not prepare for the upcoming risk of the remote team setup.

2) *Elaboration*

In the elaboration phase, the tasks are mostly related to design. Architecture is believed to heavily affect the software and the team tried to convince the client to increase the communication for the architectural review. Technical risks were identified and reported to the client regularly, which made the team feel confident about the success of the project.

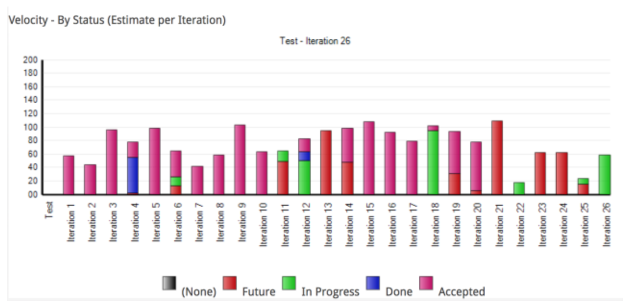
The team also established the project strategy during the elaboration phase not to repeat the mistake of the inception phase. That was mostly from what we learned in the architecture class, more specifically from Architecture-Centric Design Methodology (ACDM) [15]. We tailored the steps and procedures in accordance to our project context. The team also suggested the strategy and plan related to it. Contrary to the frustration in the elaboration phase, a well-established reporting channel and risk management strategy boosted the team morale as well as the client satisfaction.

3) *Construction*

Implementing is what software engineers enjoy and indulge in the most. The team was given 48 hours of work each week. The common working time was set up during the weekdays to work together, and a daily scrum meeting was planned to check the status. With plenty time of work, developing two features the client asked was considered not a big deal for the team.

The plan was only a plan again, though. The chronic time management problem still did not show any hint of the improvement for some members. The daily meeting was switched to two meetings per week. The quality plan and the milestones have continuously changed because the client did not respond to any reports from us. It was the last opportunity for the team to co-work because one member would be in a remote place in the next phase. The Studio project is an academic course and that aspect heavily influenced for the team members. The benefits in the elaboration phase did not last long because the reporting channel with the client was collapsed and the social skills were useless.

The best lesson the team members learned is the importance of communication with the client. We were at a loss when the client just quit the connection and became contactless from time to time. This time, we changed the policy. The customer liaison, which has already existed, notified that he/she would try to contact with several times using email, text, and phone calls. When there was no response for those efforts, the team finally made their own decisions. At least, we tried to remove some uncertainties and the team was able to deliver the mobile application



(b) Velocity Status

Figure 2. Team's velocity trend

integrating two features. That would not be exactly what the client wanted at first, but it was the minimum we could make without enough communication.

4) Transition

The team finally faced the distant team setting in the transition phase. Actually, that intrigued the team because it was not common in small and medium sized businesses. The transition phase normally does not involve many tasks related to the development, so it was not easy to assess how the suggestions for the remote collaboration would work.

Collaborating as a team can be a real challenge. Getting everybody on the same page, assigning tasks, following up on pending items, and making sure everyone is always in the loop is never easy, and it is something almost all companies struggle with. The team decided to have a weekly meeting considering the time difference and team members' schedule. The team used a videoconferencing tool like Skype or Hangout of Google to get together. The team prepared the remote development condition from the construction phase, but that was not enough. More documents were needed for the remote member to catch up. More methods to collaborate online should have been attempted.

In the early weeks of the transition phase, the weekly meetings were canceled or held without getting the whole team members. The meeting itself was not satisfactory: just checking and reporting the status without enough discussion and review of the deliverables and the iteration process. The team did not take advantage of the current collaboration technologies. Whereas the Studio project did not see the effectiveness from the remote team condition, in another situation of the remote class we took at the same time, the collaboration was good enough. The class asked for the group presentation about one topic and we were in the same distant team setting. The group shared the goals of the presentation and divided the parts each member had to do. We had a weekly meeting to check each member's progress after working individually. A subjective criterion would be the members' morale whereas an objective one would be the grade for each class. The results in the Studio project were poor in both criteria.

One way of assessing the success of the team in the agile method is the trend of team's velocity. It could be applied to the evaluation of distant team in the agile method. Comparing the velocity in the co-located situation with one in the remote condition will show the effectiveness of the team's status. The team's velocity did not show the improvement during the project in Figure 2. Overall the trend is not stable except during the elaboration phase from iteration 12 to iteration 17. Some tasks are not finished on time during the iteration in the transition phase after iteration 22.

C. Discussion

In this section, we will investigate how these suggestions would make better this situation or what were the issues when adopting these into the real situation.

1) Agile team

Identifying social skills refers to acquiring diverse communication methods both among team members and for the customer. Even though the agile method was adopted, which requires the intimate and quite often conversation, the team was too passive to just wait requirements from the customer. The team established several kinds of communication methods: a *Facebook group* between team members in addition to traditional ways such as email and instant messaging, and *biweekly teleconference meeting* with the customer.

Reporting channels are the official procedure for discussing, negotiating and satisfying the customer expectation for the project. The team did not have the on-site customer even though adopting the OpenUp. So the team needed to set up the reporting channel, and a biweekly teleconference meeting was held with the customer to report the progress of the project, and the customer gave feedback about it.

Risk management strategy is emphasized by the nature of the OpenUp requiring risk management process at the end of iteration. The team adopted the aforementioned risk overhaul in the elaboration phase. The customer wanted the team to follow feature-by-feature development for the mobile application. When we touched another one after finishing one feature, new kinds of risks were identified and the team needed to see it differently from the usual risk management process.

As these suggestions were applied to the real agile project, the problem behind them is always the motivation. When the team members are not motivated to use them, the collaboration skills are meaningless. In fact, the team was not able to build some management foundation before realizing the team's collaboration and coordination problems and raising the awareness of the importance of them.

2) Distant team

Sharing common goals and vision in the early phase of the project is the first thing we had to consider. The team had co-located setting except in the transition phase that was good enough for having the common goals and objectives. Maintaining the commonality, however, should be kept throughout the entire project when the change happens.

TABLE I
COLLABORATIVE SKILLS FOR THE STUDIO PROJECT

Skills	Team's methods	Criteria for the skills
Social skills	Email, Facebook group, biweekly teleconference meeting	Team members' and client's morale (questionnaire)
Reporting channels	VersionOne report, biweekly teleconference meeting	Number of reporting
Risk management strategy	Risk evaluation at the end of iteration Risk overhaul	Trend of the number of risks
Common goals in the early phase	Requirement engineering (RE) in co-located environment	Time spent in RE Number of requirements
Explicit roles and responsibilities	Assign of role to each member	Assigned roles
Autonomy and accountability	Distant team member management by formal (VersionOne) and informal (regular videoconferencing) method	Progress report by team member
Relate risk to problem	Risk/problem analysis	Number of problems from the risks

Explicit roles and responsibilities are a factor which enables to proceed in the distant development environment. The distant team member should be able to know what his tasks are, when they should be done, and how they can be incorporated into the deliverables of the project. It is only possible when the team defines roles and responsibilities for each team member.

Autonomy and accountability is an integral part when we deal with the team morale and the project accomplishments. In reality, it is not feasible to micromanage the distant team member. One of solutions is to give autonomy and ask accountability for the results. The team leader or project management should be able to ask for accountability for his tasks.

Risk/problem analysis is supplemental to the existing risk management process. A risk may become a problem or not, and the distant team condition may bring confliction when it becomes a problem. Without complete analysis about the reason and implications of the risk/problem, the team might evade the responsibility or accuse someone else who is not present, thus infringing collaborative team spirit.

Table 1 summarizes the suggested skills and the corresponding methods in the Studio project. The criteria for the skills are measured by both subjectively like questionnaire and quantitatively.

IV. CONCLUSION

In this paper, the trend of collaboration in software engineering was reviewed, and some suggestions were proposed for the agile process and distributed development environment. Agile process is known for strengthening the collaboration with the customer, but it is necessary to prepare strategy and procedure beforehand about how to communicate both within the teams and among the customer.

Management skills in distributed development environment presented in this paper focuses on human factors. Respecting, understanding given circumstances of each team will facilitate the collaboration. Besides, thorough preparation and planning regarding how to manage the project will drive collaborative team members to follow the practices of software engineering.

Some issues and reflections are discussed when we implemented these skills into the real software project. Our team had both characteristics of agile and distributed development. We learned that coordinating and collaborating are hard to obtain from some of experience in the project because of human and technological factors.

ACKNOWLEDGMENT

This work was supported by Electronics and Telecommunications Research Institute (ETRI) Grant funded by the Korea government [14ZC1320, Development of WoT Collaboration Service Platform based on Social Relation].

REFERENCES

[1] Ivan Mistrik, John Grundy, Andre van der Hoek, and Jim Whitehead, Collaborative Software Engineering. Springer, 2010.

[2] Ita Richardson, Valentine Caseyb, Fergal McCafferyb, John Burtonc, and Sarah Beechama, "A Process Framework for Global Software Engineering Teams," Information and Software Technology," vol. 54, pp. 1175-1191, November 2012.

[3] Helen Sharp and Hugh Robinson, "Collaboration and coor-dination in mature eXtreme programming teams," International Journal of Human-Computer Studies," vol. 66, pp. 506-518, July 2008.

[4] Barbara Kitchenham, Stephen Linkman, and David Law, "DESMET: a methodology for evaluating software engineering methods and tools," Computing & Control Engineering Journal, vol. 8,pp. 120-126, June 1997.

[5] Barbara Ann Kitchenham, "Evaluating Software Engineering Methods and Tool Part 1: The Evaluation Context and Evaluation Methods," SIGSOFT Software Engineering Notes, vol. 21, pp. 11-14, January 1996.

[6] Rob Austin and Lee Devin, Artful making: What Managers Need to Know About How Artists Work, Prentice Hall, 2003.

[7] Kent Beck and Cynthia Andres, Extreme Programming Explained, Addison-Wesley, Reading MA, 2000.

[8] Kevin Tate, Sustainable Software Development: An Agile Perspective, Addison-Wesley Professional, 2005.

[9] Ray C. Williams, George Pandelios, and Sandra Behrens, Software Risk Evaluation (SRE) Method Description (Version 2.0), SEI, 1999.

[10] Ita Richardson, Miriam Q'Riordan, and Valentine Casey, "Knowledge Management in the Global Software Engineering Environment," ICGSE 2009, pp. 367-369, 2009.

[11] Christof Ebert and Philip De Neve, "Surviving global software development," IEEE Software, vol. 18, pp. 62-69, March 2001.

[12] Matthew Bass, James D. Herbsleb, and Christian Lescher, "A Coordination Risk Analysis Method for Multi-Site Projects:Experience Report," 2009 IEEE International Conference on Global Software Engineering, pp. 31-40, November 2009.

[13] Mary Shaw, Jim Herbsleb, Ipek Ozkaya, and Dave Root, "Deciding What to Design: Closing a Gap in Software Engineering Education," ICSE 2005, pp.607-608, May 2005.

[14] Eclipse Foundation, OpenUp Wiki, http://epf.eclipse.org/ wikis/openup, 2014.08.15.

[15] Anthony J. Lattanze, Architecting Software Intensive Systems, Auerbach Publications, 2009.