2009 Fourth IEEE International Conference on Global Software Engineering

How Technological Support Can Enable Advantages of Agile Software Development in a GSE Setting

Kevin Dullemond
Delft University of Technology
k.dullemond@tudelft.nl

Ben van Gameren
Delft University of Technology
b.j.a.vangameren@tudelft.nl

Rini van Solingen
Delft University of Technology
d.m.vansolingen@tudelft.nl

Abstract

Because of the distance between the dispersed development locations, Global Software Engineering (GSE) is confronted with challenges regarding communication, coordination and control of the development work. At the same time, agile software development is strongly built upon communication between engineers and has proven its benefits, although, mostly on one single site. As such, it might be advantageous to combine GSE with agile development. This blend however is not straightforward since the distributed and agile development approaches might have conflicting convictions. In this paper we will discuss the advantages and challenges of combining GSE with agile development based on a theoretical, literature-based research. The main results presented in this paper are: (i) aspects of agile software development, (ii) benefits and challenges associated with these in relation to GSE, (iii) categories of technological support for agile GSE and (iv) a framework depicting the mutual relations among them.

1 Introduction

Global Software Engineering (GSE) is becoming increasingly interesting due to the globalization of business [13, 32, 21, 34, 48, 2]. In GSE the software development process is distributed between several geographically dispersed locations [20, 21, 50]. Advantages of GSE include: market-proximity [28, 31, 21], reducing time-to-market by working around the clock [13, 29, 24, 21], flexibility with respect to business opportunities [13, 29], reducing costs by delegating work to countries with low labor cost [14, 21] and being able to fully utilize available resources [32, 28, 21]. Besides being beneficial, GSE introduces a number of challenges in relation to communication, coordination and control of the development process [14]. Examples are: lack of informal communication [13, 29, 32, 3], reduced hours of collaboration [7, 40, 39, 2], communication delay [3, 33, 34, 20] and loss of cohesion [13, 30, 34]. All these challenges originate from the existence of three kinds of distances: geographical, temporal and socio-cultural [13, 43, 20]. The combination of these distances is what makes GSE complex [38]. A way to deal with the challenges commonly faced in GSE would be to a. reduce these distances themselves b. reduce the consequences of the existence of these distances or c. help to cope with the consequences of these distances. It is expected this can be achieved by incorporating certain aspects of agile software development into GSE [38, 45]. In this paper we analyze agile development to determine how it can improve upon GSE and how this can be supported with technology.

The core question in this research is: *"What are the advantages and challenges of the combination of agile software development and GSE, and how can these be supported with technological aid?"*

In section 2 we define a subdivision of agile software development into aspects. With these aspects we systematically discuss, in section 3, how agile software development can be beneficial explicitly with respect to GSE, and we also discuss how carrying out these aspects can be more difficult in a distributed setting. Subsequently, in section 4 we define five categories of technological support which are useful to agile GSE. In section 5 we discuss how to support the incorporation of the aspects into GSE best. Finally we summarize our findings in a framework, reflect on our work and present possibilities of further research.

2 Aspects of agile software development

In this paper we are concerned with how agile software development can be beneficial in the context of GSE. Therefore we define a subdivision of agile software development into aspects to be able to systematically discuss how agile software development can be beneficial explicitly with respect to GSE. We derived these aspects from

978-0-7695-3710-8/09 $25.00 © 2009 IEEE
DOI 10.1109/ICGSE.2009.22

the practices and guidelines mentioned in literature. Many sources report the development of lightweight methodologies [25, 37, 1, 19, 54, 5]. Examples of these new methodologies are: Scrum [51, 52], eXtreme Programming [9, 8, 27, 10], Crystal methods [17, 18], Adaptive Software Development [35, 36, 37] and Feature Driven Development [16, 46]. After the development of these new methodologies, seventeen prominent process methodologists formed the *"Agile Alliance"* and wrote *"The Agile Manifesto"* in which they defined a set of principles [4]:

- AM_1 *Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.*
- AM_2 *Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.*
- AM_3 *Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.*
- AM_4 *Business people and developers must work together daily throughout the project.*
- AM_5 *Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.*
- AM_6 *The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.*
- AM_7 *Working software is the primary measure of progress.*
- AM_8 *Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.*
- AM_9 *Continuous attention to technical excellence and good design enhances agility.*
- AM_{10} *Simplicity–the art of maximizing the amount of work not done–is essential.*
- AM_{11} *The best architectures, requirements, and designs emerge from self-organizing teams.*
- AM_{12} *At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.*

The aspects derived in this section are in fact generalizations of these practices and guidelines. With the term *aspect of agile software development* we denote the goals agile software development attempts to accomplish. More explicitly, for the Agile Manifesto and agile methodologies which define explicit practices, these aspects can be seen as the outcome of using an agile practice from a certain subset of agile practices. We will explain each of these aspects by giving a definition of what the aspect entails followed by a clarification of how exactly the aspect is reflected in the Agile Manifesto.

A_1 Close collaboration among the members of the development team

A_1 concerns that all members of the development team should work together daily throughout the project in a very close way, while communicating frequently. This aspect is reflected in the Agile Manifesto in the following way: For one it urges that business people and developers should work together daily throughout the project (AM_4). Also, it states that the best architectures, requirements and designs emerge from self-organizing teams (AM_{11}). For teams to be self organizing the members need to work together closely. Lastly, it claims the most effective and efficient method of conveying information to and within the development team is face-to-face conversation (AM_6).

A_2 Short iterations, frequent builds and continuous integration

A_2 concerns that development is done by delivering incremental components of business functionality in so-called iterations. Developing a product in this fashion helps to keep focus on short term goals and to create working software quickly. During these iterations the work should be integrated and build as frequent as possible to be able to detect and resolve problems early. After each iteration ends and before the next one begins, the process that is used should be adapted to work even better in the next iteration [25]. The processes associated with each of the agile methodologies mentioned earlier concern a development process consisting of short iterations with frequent builds. While the Agile Manifesto does not explicitly specify a process, it does state that the highest priority is to satisfy the customer through early and continuous delivery of valuable software (AM_1), that software needs to be delivered frequently, with a preference to a short time scale (AM_3), and that working software is the primary measure of progress (AM_7). It also states that the team should reflect, at regular intervals, on how to become more effective and then tune and adjust its behavior accordingly (AM_{12}); thus promoting self-adaptivity of the process. Finally continuous integration is closely related with receiving feedback, since by continuously integrating and building, feedback can be acquired more often. This regards both feedback acquired from colleagues and the customer as well as feedback acquired from the results of tests. Feedback is reflected in the Agile Manifesto, as can be gathered from principle AM_9, which subscribes the continuous attention to technical excellence and good design.

A_3 Decentralizing the decision making

In agile software development part of the decision making is moved to the developers. Management is still needed to remove roadblocks standing in the way of progress. The development team, however, is entitled to make certain technical decisions without involvement of the management. This aspect also concerns the decisions made by a subgroup of a development team rather than an individual. This aspect is less directly found in the researched methodologies. Most agile methodologies however, advise having faith in the abilities of the developers. The Agile Manifesto for instance states that projects should be built around motivated individuals that receive the environment and support they need and that they should be trusted to get the job done (AM_5). It also emphasizes the power of self-organizing teams (AM_{11}) and that such teams need to be trusted to get the job done. This is exactly what this aspect seems to be about most; management should recognize the expertise of the developers [25].

A_4 **Customer involvement**

A_4 involves the active participation of the customer in the development of the system. This involvement is used to acquire feedback on the current implementation of the system and to further clarify the requirements of the system to be built. It is important to note that the customer has the right to change requirements during the entire project. The Agile Manifesto emphasizes the importance of the customer in the development process. Again we refer to the fact that the Agile Manifesto regards satisfying the customer through early and continuous delivery of valuable software as having the highest priority (AM_1). Also it states that changing requirements should be welcomed, even late in development (AM_2). Besides this also in the values of the Agile Manifesto the importance of this aspect can be seen, as these both mention the importance of customer collaboration and responding to change.

A_5 **Collective ownership of work**

A_5 concerns that what is produced is the result of the entire team and not of an individual. No single team member owns, or is responsible for a specific code segment and all work can be changed by the entire team, without explicit permission. This aspect, however, also concerns the development team having a shared vision and responsibility of the system to be built. The whole team is creating something together, aiming for a single goal and is collectively responsible that this goal is reached. This aspect is found in a indirect manner in the researched methodologies. Chao et al. [15] states: *"... several agile methods (e.g. XP and Scrum) imply that explicit knowledge including designs and models should be collectively owned"*. It is true that this aspect is merely implied in the Agile Manifesto. The Agile Manifesto states that the best architectures, requirements and designs emerge from self-organizing teams (AM_{11}) and these kind of teams are teams with a shared responsibility with respect to the work they are performing. The principle which states projects should be built around motivated individuals who are given the environment and support they need and are trusted to get the job done (AM_5) implies the project is the shared responsibility of the entire development team.

A_6 **The system to be built is most important**

A_6 concerns that working software is the most important goal of the project and other matters, like documentation, are inferior to it. Spending a lot of time on updating non-essential artefacts should be prevented. One of the values of the Agile Manifesto states *"working software over comprehensive documentation"* which is exactly what this aspect is about. This aspect is also reflected in two of its practices. Firstly it states working software is the primary measure of progress (AM_7) and secondly it states the highest priority is to satisfy the customer through early and continuous delivery of valuable software (AM_1).

A_7 **Favoring simplicity**

Agile software development favors simplicity, or the maximization of the amount of *work not done* [4]. This aspect concerns looking for the simplest working solution first and improving it later only if the need arises. This aspect is based on the assumption that requirements and other matters with respect to the project are likely to be changed in the future so that it is rarely worth to implement things supposed to be helpful at some point in the future. The relation of this aspect to the Agile Manifesto is rather direct since it states that it is essential (AM_{10}).

A_8 **Sustainable pace of development**

A_8 is concerned with the fact that people should not work excessive overtime for long periods of time. The motivation behind this is that un-energized people produce less and lower quality work than energized people. DeMarco states: *"Extended overtime is a productivity reducing technique"* [22]. The idea is that the development speed is such that all people involved in the project are able to maintain a constant pace indefinitely. Again the relation to the Agile Manifesto is rather direct since it prescribes that the sponsors, developers and users should be able to maintain a constant pace indefinitely (AM_8).

Overview

In table 1 an overview is given of the relation between the principles of the Agile Manifesto and the aspects of agile software development we defined in this section. In this table it can be seen that each of the practices of the Agile Manifesto is covered by at least one of the aspects we defined., thus increasing the likeliness that the list is complete. For a similar comparison with eXtreme programming we refer to [23].

Practice	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8
AM_1		X		X		X		
AM_2				X				
AM_3		X						
AM_4	X							
AM_5			X		X			
AM_6	X							
AM_7		X				X		
AM_8								X
AM_9		X						
AM_{10}							X	
AM_{11}	X		X		X			
AM_{12}		X						

Table 1. Relation between the Agile Manifesto and the agile aspects

3 Benefits and Challenges of incorporating agile aspects in GSE

When carrying out software development projects professionally, a process is used to prevent the project from lapsing into chaos. What kind of process should be used depends on the specific requirements and constraints of the project. Barry Boehm argues: *"Both agile and plan-driven methods have a home ground of project characteristics in which each clearly works best, and where the other will have difficulties. Hybrid approaches that combine both methods are feasible and necessary for projects that combine a mix of agile and plan-driven home ground characteristics."* [12]. For globally distributed projects however, Grinter et al. [28] suggest using a structured, plan driven process, as a way to coordinate such projects. Nevertheless there are two reasons why less plan-driven, more agile practices could be useful for GSE projects:

1. They can be useful in the same way they are useful to software development in general. This regards that in general, they are able to better cope with uncertainty and changing requirements in projects than plan-driven approaches.
2. They can be useful with respect to GSE in particular by reducing the negative influence of distance on [14]: communication, coordination and control.

In this paper we are concerned with reasons of the second category, which we will call *benefits*. For each aspect we will discuss the benefits it has with respect to each of the three distances. Besides being beneficial to GSE, agile aspects can also be more difficult to incorporate correctly because of the distances since the original methodologies these are derived from were not created to deal with these [41, 45]. We will call such problems *challenges* and will also discuss these for each aspect.

A_1 Close collaboration among the members of the development team

Close collaboration leads to good report between colleagues. Good report between colleagues means they are more willing to compromise with respect to working times (B_1). An example of this effect with respect to XP pair programming is mentioned by Agerfalk et al.: *"... people were flexible and, even though there was a delay in response, individual developers tried hard to spend as much time as possible with the distributed pair programmer"* [38]. Hence the negative influence of the *temporal* distance is reduced since the time overlap is increased in comparison to both team members just working normal office hours and not being flexible. The main challenges which are a direct consequence of the *geographical* distance in GSE are the lack of informal communication and the increased effort to initiate contact. At the same time close collaboration between team members generally increases informal communication and eases initiating contact (B_2). Therefore we can say close collaboration between team members *reduces* the main consequences of the geographical distance. Close collaboration, does not just lead to good report between colleagues, it also leads to a better mutual understanding between them (B_3). Therefore close collaboration actually reduces the *socio-cultural* distance between team members.

The *geographical and temporal* distances between developers in GSE projects causes frequent communication between developers to be difficult (C_1) [53, 26, 49, 45]. Because of this, close collaboration is also harder to accomplish than when the development team is collocated since frequent communication is essential for close collaboration. In distributed development, participants at different sites are less likely to perceive themselves as part of the same team than with collocated development due to the *socio-cultural* distance (C_2). Because of this, team members find it hard to have faith in the good intentions of remote colleagues [41] and there can be a lack of team cohesion [49]. These things make close collaboration between developers at different geographical locations more difficult.

A_2 Short iterations, frequent builds and continuous integration

Applying short iterations, frequent builds and continuous integration in the development process leads to feedback. This feedback motivates developers [45] and motivated developers feel more like a team (B_4). Next to this, seeing high quality work early and frequently results in trust (B_5) [45]. Both feeling more like a team and increased trust reduce the *socio-cultural* distance caused by working globally distributed. In GSE projects in general, it is quite hard to have a good idea of the progress of work, mainly due the *temporal and geographical* distances which make direct communication difficult. Because of this, problems can be detected too late, and this can result in time and resources being wasted. Continuous integration can improve upon this situation because it leads to transparency of the progress (B_6) [53, 45]. Another issue in GSE projects aggravated by *geographical and temporal distance* is configuration management: the integration of the complete system. Continuously integrating, in contrast to integrating everything at the end of the project, makes configuration management less of an issue (B_7) [53].

Performing short iterations, frequent builds and continuous integration is harder in a distributed than in a collocated setting for a couple of reasons. For one, the lengths of the iterations should be increased to compensate for the time lost due to communication overhead (C_3) [26, 44]. Secondly, iterations using an agile development approach often define meetings at the start of the iteration and regular short status meetings during the iterations. It seems wise to carry these practices over to the distributed environment for proper co-

ordination between teams but this is also difficult (C_4), both due to the *temporal* distance and because it is often impossible to perform such meetings in a single room due to the *geographical* distance between the teams. Lastly, configuration and version management is also more difficult in a distributed setting [45]. Even though teams at different *geographical* locations are working on the project, possibly at the same time, an unambiguous global view of the current system needs to be maintained (C_5). To accomplish this, a common code base should be used [47, 45]. This can be difficult because of the *geographical* distance, since the common code-base must be located somewhere. Another reason the unambiguous global view can be difficult to be maintained is the *temporal* distance, since colleagues are not always available when needed.

A_3 Decentralizing the decision making

Because of this aspect, both the *geographical and temporal* distances become less of an issue since, developers can take certain decisions without having to confer with management; which could be located in another part of the world (B_8). Next to this, the *geographical and temporal* distances often make it hard to make proper estimations regarding the progress of the project. By letting the project team make these estimations they are often more accurate (B_9) [53]. Giving the individual developers and development teams autonomy is a great motivator and allows people to be more productive [6, 26]. As mentioned earlier, this increase in motivation leads to developers feeling more like a team (B_{10}), which reduces the *socio-cultural* distance between them. Next to this, allowing teams and individual developers to make their own decisions conveys trust (B_{11}). According to Zand [55], one of the most powerful ways to show your own trustworthiness is to trust the other.

In order to use decentralization of decision making properly within a project, the developers should be autonomous and independent enough to actually make decisions on their own. This can be a problem because people are often trained to listen to their superiors and not make decisions on their own (C_6) [6, 26]. Furthermore it can be hard to make decisions decentralized when *geographically and temporally* separated from the rest of the development team. This is because the developer which has to make the decision can reach better decisions when he has access to certain types of information, which can be harder to acquire in a distributed setting (C_7).

A_4 Customer involvement

In the description of A_2, we described that constant feedback decreases the *socio-cultural* distance between team members, because feedback motivates the developers and seeing high quality work early and frequently builds trust. Feedback, however, is much more valuable when it is given by the actor for which the system is being developed: the customer (B_{12}).

In distributed development, the customers may be located far away making frequent interaction more difficult to arrange (C_8). Having the customer on site is even harder in this situation and if it can be arranged the on-site customer might gradually lose connection to his or her own environment (C_9) [45].

A_5 Collective ownership of work

Carmel defines several characteristics which a real team should posses [13]. Two of these are: *"collective responsibility for its products"* and *"shared responsibility for managing its work"*. Collective ownership reflects both these characteristics so it will lead to closer development teams (B_{13}), which, in turn, reduces the *socio-cultural* distance between its members.

The main problem with incorporating this aspect of agile methodologies in a GSE context is that the very idea might conflict with certain ideologies and cultural believes (C_{10}). Berteig [11], for instance, discusses teaching a course on agile development in Romania. Several of the participants felt that the collective ownership of work closely resembled communism, to which they objected. Another problem with incorporating this aspect in a distributed setting is that all work should be accessible by everyone from anywhere in order for all the work to truly be collectively owned (C_{11}).

A_6, A_7 and A_8

A_6, A_7 and A_8 do not offer specific benefits when working globally distributed compared to working co-located. Therefore there are no associated benefits and challenges specific to the GSE environment.

Overview

In table 2 and 3 an overview is given of the benefits and challenges of agile GSE.

Aspect	Distance	Description
A_1	Temporal	B_1: More overlap in working time
	Geographical	B_2: Increase in informal communication
	Socio-cultural	B_3: Better mutual understanding
A_2	Socio-cultural	B_4: Feedback motivates the developer
		B_5: Seeing high quality work early and frequently results in trust
	Geographical and Temporal	B_6: The process is more transparent
		B_7: Configuration management is less of an issue
A_3	Geographical and Temporal	B_8: Less control needed
		B_9: Better progress estimations
	Socio-cultural	B_{10}: Autonomy motivates the developers
		B_{11}: Being trusted results in trust
A_4	Socio-cultural	B_{12}: The customer can give valuable feedback
A_5	Socio-cultural	B_{13}: Increase in team cohesion

Table 2. Benefits of agile GSE [23]

Aspect	Distance	Description
A_1	Geographical and Temporal	C_1: Frequent communication is difficult
	Socio-cultural	C_2: Team members are less likely to perceive themselves as part of the team
A_2	All	C_3: Less can be achieved in a certain amount of time due to communication overhead
	Geographical and Temporal	C_4: Stand-up and status meetings are more difficult
		C_5: Unambiguous global view of the current system is more difficult to be maintained
A_3	Socio-cultural	C_6: Some developers might not be autonomous and independent enough to make decisions themselves
	Geographical and Temporal	C_7: Having access to sufficient information in order to reach correct decisions is more difficult
A_4	Geographical and Temporal	C_8: Frequent interaction with the customer is difficult
		C_9: When the customer is placed on-site, he might gradually lose connection with his own environment
A_5	Socio-cultural	C_{10}: People might object to the notion
	Geographical and Temporal	C_{11}: It is more difficult for all work to be accessible to everyone at any time from any location

Table 3. Challenges of agile GSE [23]

4 Supporting agile GSE with technology

In the previous section we discussed the benefits and challenges associated with agile GSE. This paper is concerned with how to support these with technological aid[1]. In order to properly discuss how to support agile GSE with technology, different types of technological support, beneficial in that specific environment, should be defined. We will define these categories based on what requirements are needed in supporting technology for agile GSE. They are strongly based on the five objectives of supporting technologies for collaborative work, defined by Carmel [13].

The first difficulty that can be supported with technology is the communication between team members (R_1). This is required because the software development process does not operate on mandate, orders and edicts but on the self-managing capabilities of the developers [13, 42]. Hence the technology needs to provide an infrastructure for collaborative sessions, making it easier to interact with colleagues, in order to support this self-managing character of software development [13, 42]. Another problem with GSE is the risk of *re-inventing the wheel*. This risk can be reduced when tools are used to create a rich and up-to-date *project memory* (R_2) including versioned files, change histories and technical documentation [13, 42, 34]. Thirdly, in a GSE project it is much harder to understand what other project members are doing than in a co-located project [13, 42, 34]. Therefore project transparency is necessary in order to coordinate effectively with the members of the team. Technology which provides status information about tasks, people and other dynamic team information to all team members at different sites can increase the transparency of the project (R_3). Fourthly, in a distributed setting quality assurance is especially important to monitor and guarantee the quality of the product. Technology which supports quality assurance functions, such as functional testing can have a positive influence on the total quality of the product (R_4) [13]. Up until this point only technological support which support GSE in general have been discussed. We conclude with a requirement category specifically applicable to agile GSE: technology facilitating continuous integration and frequent builds (R_5) [23]. Examples of such technologies are automated build systems and configuration management tools. Below the requirement categories discussed are summarized [13, 42, 34, 23]:

- R_1 *Facilitates direct contact between colleagues*
  Technological support which facilitates direct communication between two or more actors.
- R_2 *Facilitates knowledge sharing among colleagues*
  Technological support which facilitates the sharing of *technical* project knowledge.
- R_3 *Facilitates transparency of the project status*
  Technological support which facilitates the sharing of *organizational* project knowledge.
- R_4 *Facilitates quality assurance*
  Technological support which facilitates quality assurance functions to monitor and guarantee the quality of the product.
- R_5 *Facilitates continuous integration and frequent builds*
  Technological support which eases the process of continuously integrating the system as well as producing builds frequently.

Of these requirements of technologies, R_1 through R_3 can often be achieved by the same tools since these categories all have to do with communication. Tools can, however, be more appropriate to support one category than another. Instant messengers and conferencing software are for example suitable to category R_1 while wikis and tracking software are more appropriate to R_2 and R_3. Tools which support categories R_4 or R_5, on the other hand, are often specific to just one of these two categories. Finally there exist technologies which attempt to integrate several of the requirements into a single environment. Examples of these, are tools like Jazz, Merlin Toolchain and Microsoft Visual Studio Team System which attempt to extend Integrated Development Environments (IDEs) with collaborative functionalities. In [23], a more extensive discussion regarding existing supporting technologies and their relation with the requirement categories is presented.

5 How each aspect can be supported by different types of technology

In order to incorporate the five aspects of the agile development approach, that influence distance, into the GSE development process, in the best possible way, the challenges faced should be alleviated and the benefits should be exploited. In this section, we discuss what types of technological support are beneficial to which aspects or more precisely: which challenges and benefits belonging to the aspects can be supported by which type of technological aid. To do this we have presented tables 2 and 3 in a different manner in figure 1, showing the relationship between

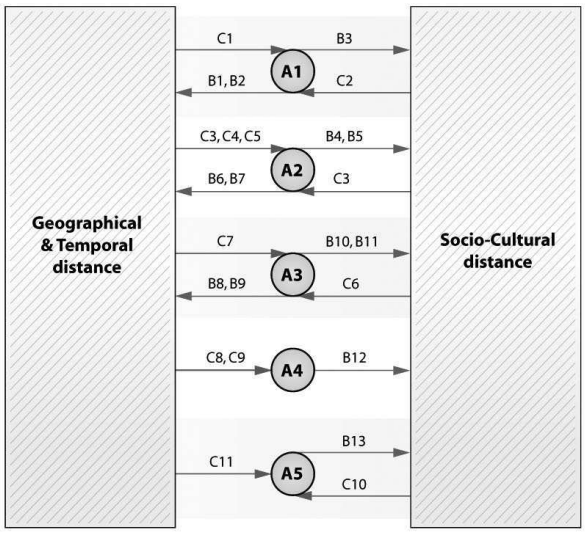
[1] See [23] for ways to support the challenges with non-technological aid

the aspects and the distances. In this figure, the distances and aspects are depicted as nodes and the interdependencies as directed arcs, which correspond to either a benefit or a challenge. The arcs which correspond to a benefit are directed from the aspect to the distance it targets and represent how the aspect helps to cope with this distance. The arcs which correspond to a challenge are directed from the distance which causes the challenge to the aspect that the challenge opposes, and represent how the distance makes it harder to carry out the aspects.

This representation of the relation between the aspects and the distances shows that the aspects can be supported directly by supporting their associated benefits and challenges, or indirectly through other benefits and challenges. Next, for each aspect we will discuss how each of its benefits and challenges can be supported by the requirement categories defined in the previous section.

A_1 Close collaboration among the members of the development team

The main challenge with respect to aspect A_1 is challenge C_1: the fact that frequent communication is difficult in a distributed setting. Because of this, close collaboration among team members is more difficult. This can be alleviated by technological support, which makes it easier for team members to communicate and by technological support which eases knowledge sharing in the entire development team; categories R_1 and R_2 respectively. Features belonging in category R_3 could also help to initiate contact, if the specific feature makes it possible to determine who to contact by offering information with respect to individual knowledge and availability of colleagues. In this particular case, the same types of technological support that help to deal with challenge C_1 also support benefit B_2 because C_1 and B_2 are closely related.



A1	Close collaboration among the members of the development team
A2	Short iterations, frequent builds and continuous integration
A3	Decentralizing the decision making
A4	Customer involvement
A5	Collective ownership of work
B1	More overlap in working time
B2	Increase in informal communication
B3	Better mutual understanding
B4	Feedback motivates the developer
B5	Seeing high quality work early and frequently results in trust
B6	The process is more transparent
B7	Configuration management is less of an issue
B8	Less control needed
B9	Better progress estimations
B10	Autonomy motivates the developers
B11	Being trusted results in trust
B12	The customer can give valuable feedback
B13	Increase in team cohesion
C1	Frequent communication is difficult
C2	Team members are less likely to perceive themselves as part of the team
C3	Less can be achieved in a certain amount of time due to communication overhead
C4	Stand-up and status meetings are more difficult
C5	Unambiguous global view of the current system is more difficult to be maintained
C6	Some developers might not be autonomous and independent enough to make decisions themselves
C7	Having access to sufficient information in order to reach correct decisions is more difficult
C8	Frequent interaction with the customer is difficult
C9	When the customer is placed on-site, he might gradually lose connection with his own environment
C10	People might object to the notion
C11	It is more difficult for all work to be accessible to everyone at any time from any location

Figure 1. Complete overview of relationships aspects and distances [23]

A_2 Short iterations, frequent builds and continuous integration

With respect to this aspect various challenges and benefits can be alleviated or exploited respectively, by means of technological support. For one, challenge C_5 can be alleviated both by facilitating knowledge sharing and facilitating transparency of the project status. This is because knowledge sharing will provide team members with a better and more complete understanding of the technical project knowledge. The transparency of the project status, on the other hand, will ease the access to the knowledge regarding how the project is to be carried out and how this is progressing. Both these types of information together make up an unambiguous global view of the current system. Technological support from categories R_2 and R_3 alleviate access to these types of information and thus alleviate challenge C_5. Another challenge which can be alleviated by means of technological support is challenge C_4. By facilitating direct contact, having meetings geographically distributed is made easier. Therefore technological support from category R_1 will directly target this challenge. Another factor to consider when dealing with this challenge is the difficulty to arrange the meetings when the people required to be present do not share the same working hours. By facilitating access to the information regarding the availability of the participants, as well as other relevant data, this challenge can be directly dealt with. Technological support from category R_3 accomplishes this. Lastly, technological support from category R_2 will also help dealing with this challenge, although in a less direct manner, because when knowledge sharing is improved, the actual need for having meetings is decreased. The last challenge associated with this aspect, challenge C_3, can be supported by technology from categories R_1, R_2 and R_3 because these all facilitate a certain

aspect of communication and when communication is less difficult, the communication overhead will be decreased. Next we will discuss how the benefits caused by this aspect can be exploited by means of technological support. Firstly benefit B_4, can be exploited by facilitating quality assurance, technology from category R_4, because better quality assurance will improve the quality of the feedback. Technological support from categories R_1, R_2 and R_5 helps to exploit this benefit in an indirect manner. Support from R_5 causes the feedback to be more up-to-date because the information the feedback is based on is created more often. Support from R_1 and R_2, finally, helps exploit this benefit by helping to make the feedback accessible to the appropriate people. Benefit B_5 is related to benefit B_4 in the sense that feedback can also concern the quality of work. In this benefit however, emphasis lies on team members witnessing the high quality themselves. This can be achieved, either by examining a prototype of the system or by taking note of certain measures which increase the chances of creating a high quality product. An example of such a measure is employing a certain testing scheme. When this is done and the results of these tests are positive, this is an indicator that the work is of high quality. These ways to witness the high quality of work can be supported by technology offering features from categories R_4 and R_5. Technology from R_4 offers measures to assure that a high quality system is created and because of support from R_5 it is easier to have access to a prototype of the current system. Another benefit caused by aspect A_2 is benefit B_6, which can be supported by category R_3 because this provides an overall view of the current state of the project and thus increases the transparency of the project status directly. Technology from R_4 and R_5 is also useful since R_4 enables the quality aspect of the progress to be available and R_5 enables the overview to be updated more often and thus to more accurately reflect the actual state of the project. The last benefit of aspect A_2, benefit B_7, concerns configuration management being easier because of continuous integration. This can be supported by technology which eases the integration of the portion of the system that is complete. This is achieved by technology from category R_5.

A_3 **Decentralizing the decision making**

With respect to this aspect, challenge C_7 can be alleviated by use of technological support by easing the access to the knowledge necessary to reach the right decisions. Because this mainly concerns technical decisions, technological support from R_2 can be used to directly alleviate the challenge, while support from R_3 is useful as well, yet in an indirect manner. This is because knowledge regarding the organizational status and setup of the project will provide a context to make the decisions in, yet the information the decisions are mainly based on will be technical information. The same is true with respect to the exploitation of benefit B_8 since when better decisions are made, less control is needed in both directions between the people technically implementing the project and the people managing the project. Benefit B_9 is further exploited directly by technological support from both R_2 and R_3. This is because both knowledge regarding the content of work to be performed and the context this work is to be performed in, directly influences the estimation of the current and expected progress.

A_4 **Customer involvement**

With respect to this aspect challenge C_8 can be alleviated in a similar fashion as challenge C_1 from aspect A_1. Thus technological support from categories R_1 and R_2 support it directly because this eases communication and R_3 supports it indirectly since this only helps with the arrangement of direct contact. Benefit B_{12} can be further exploited by facilitating for constant integration and frequent builds because this will more frequently provide the customer with prototypes, to give feedback on. Because of this, the customer can be involved without actually having to be on site, since the prototype can be sent to a remote location. This is also why technical support from categories R_1 and R_2 also helps further exploitation of this benefit, since if both the feedback and the information regarding and including the prototype are as easily accessible to the customer as possible; this will result in the best and most feedback possible.

A_5 **Collective ownership of work**

Aspect A_5 can only be supported by technological support which alleviates challenge C_{11}. This can be done by using technological support from R_2, because support from this category eases access to technological project information and so also to all work produced by the development team. Technological support from category R_3 can support challenge C_{11} in an indirect fashion, because working together in a single work-base can be done more effectively and efficiently when knowledge regarding the current and past activities of the other team members, is available.

6 Conclusions

In this paper we have deducted a set of requirements for technological support which can be used as a reference for combining GSE and agile software development. This is done through the identification of: a set of agile aspects relevant to GSE, benefits and challenges which arise when applying these in a GSE setting, and categories of technological support which are useful to agile GSE. This set of requirements can be used to address specific benefits and challenges of incorporating agile aspects by selecting or developing dedicated technology. Or in other words: the development of software tools for GSE can now directly support these benefits and challenges by fulfilling the associated requirements. The benefits and challenges can then be used to further refine the requirements for the specific features in these tools.

In table 4 a framework is presented which summarizes how technological support from each of the categories is beneficial with respect to each of the benefits and challenges. In this framework a double-plus sign indicates it is possible to use technological support from the corresponding category to directly exploit or alleviate a benefit or challenge respectively. A single-plus sign also indicates that exploitation or alleviation is possible with the help of technological support from the corresponding category. However, either the technology does not alleviate the challenge itself or the level of exploitation of the benefit, the technology can achieve, is limited. With this framework it becomes feasible to identify technological support for GSE which respectively exploits or alleviates the benefits and challenges of incorporating agile aspects.

This paper started with the core question: *"What are the advantages and challenges of the combination of agile software development and GSE, and how can these be supported with technological aid?"* The first part of this question is answered by table 2 and 3 which respectively list benefits and challenges and their relation with agile aspects and distance. The last part of this question is answered by table 4, which lists the requirements for GSE technology when incorporating the agile aspects into GSE.

There are however some limitations regarding this research. Firstly, the challenges and benefits are not mutually comparable with respect to their importance. This is because the importance of each of the challenges and benefits is likely to depend on each specific practical setting. Secondly, we cannot fully guarantee the completeness of all defined benefits, challenges, requirement categories and agile aspects, because it is impossible to derive them in a way that is both conclusive and exclusive. In other words; it is possible that there are more challenges, benefits, requirement categories and agile aspects we did not yet elicit. Thirdly, we perform categorizations which posses an inherent subjective characteristic. This applies to both the assignment of agile principles to the agile aspects and the assignment of requirement categories to benefits and challenges. This can be improved by validating the current categorizations, for example by using an expert committee. Finally, we focused on how the incorporation of agile aspects influences *distance*, either in a positive or negative way, via the associated benefits and challenges. Because of this, we have only shown the influence aspects have on each other by influencing the distances. This is not the entire story because the aspects also influence each other directly. An example is that aspect A_2, *Short iterations, frequent builds and continuous integration*, causes the more frequent building of an intermediate system and aspect A_4, *Customer involvement*, directly benefits from this because it allows the customer to give feedback, on the most recent build, more frequently. Such direct interactions between separate aspects have not been investigated.

This research has been limited to a theoretical analysis of available literature and deduction of theoretical aspects, benefits, challenges and requirement categories. These are fully based on literature and deductions from this literature. As such our findings are well-founded in theory and traceable to its sources. However, our research results have not yet been validated as a whole. As such, we will continue with the validation of these findings by means of a series of industrial case studies. These are expected to complete and/or confirm (parts of) our findings and extend the work with guidelines on how to exploit them in practice.

Aspect	Challenge/Benefit	R_1	R_2	R_3	R_4	R_5
A_1	C_1	++	++	+		
	B_2	++	++	+		
A_2	C_3	++	++	++		
	C_4	++	+	++		
	C_5		++	++		
	B_4	+	+		++	+
	B_5				++	++
	B_6			++	+	+
	B_7					++
A_3	C_7		++	+		
	B_8		++	+		
	B_9		++	++		
A_4	C_8	++	++			
	B_{12}	+	+			++
A_5	C_{11}		++	+		

Table 4. Framework of technological support for agile GSE

References

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. *Agile software development methods. Review and analysis.* VTT Publications, 2002.

[2] P. Ågerfalk, B. Fitzgerald, H. Holmström Olsson, and E. Conchúir. Benefits of global software development: The known and unknown. In Q. Wang, D. Pfahl, and D. Raffo, editors, *ICSP*, volume 5007 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 2008.

[3] P. Ågerfalk, B. Fitzgerald, H. Holmström Olsson, B. Lings, B. Lundell, and E. Conchúir. A framework for considering opportunities and threats in distributed software development. In *DiSD'05*, pages 47–61, August 2005.

[4] Manifesto for agile software development. http://agilemanifesto.org.

[5] M. Awad. *A Comparison between Agile and Traditional Software Development Methodologies*. Honours program thesis, University of Western Australia, 2005.

[6] J. Barker. Tightening the iron cage: Concertive control in self-managing teams. *Administrative Science Quarterly*, 38(3):408–437, 1993.

[7] R. Battin, R. Crocker, J. Kreidler, and K. Subramanian. Leveraging resources in global software development. *Software, IEEE*, 18(2):70–77, Mar/Apr 2001.

[8] B. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[9] K. Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[10] K. Beck. *Extreme Programming Explained: Embrace Change, Second edition*. Addison-Wesley, 2004.

[11] M. Berteig. *Agile is Not Communism*. PhD thesis, 2007. http://www.agileadvice.com/archives/2007/07/agile_is_not.html.

[12] B. Boehm. Get ready for agile methods, with care. *IEEE Computer*, 35(1):64–69, 2002.

[13] E. Carmel. *Global software teams: collaborating across borders and time zones*. Prentice Hall, 1999.

[14] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *IEEE Softw.*, 18(2):22–29, 2001.

[15] T. Chau, F. Maurer, and G. Melnik. Knowledge sharing: Agile methods vs. tayloristic methods. In *WETICE'03*, pages 302–307. IEEE Computer Society, 2003.

[16] P. Coad, E. Lefebvre, and J. De Luca. *Java Modeling in Color with UML*. Prentice Hall, 1999.

[17] A. Cockburn. *Writing effective use cases, The crystal collection for software professionals*. Addison Wesley, 2000.

[18] A. Cockburn. *Agile software development*. A-Wesley, 2002.

[19] D. Cohen, M. Lindvall, and P. Costa. An introduction to agile methods. *Advances in Computers*, 62:2–67, 2004.

[20] E. Conchúir, H. Holmström Olsson, P. Ågerfalk, and B. Fitzgerald. Exploring the assumed benefits of global software development. *ICGSE'06*, pages 159–168, Oct. 2006.

[21] D. Damian and D. Moitra. Guest editors' introduction: Global software development: How far have we come? *Software, IEEE*, 23(5):17–19, Sept.-Oct. 2006.

[22] T. DeMarco. *Slack, Getting Past Burnout, BusyWork, and the Myth of Total Efficiency*. Broadway Books, 2002.

[23] K. Dullemond and B. Gameren van. *Technological support for distributed agile development*. Master, Delft University of Technology, 2009. http://www.dullemond.co.cc/thesis.pdf.

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

[25] M. Fowler. The new methodology. *Dev. magazine*, Dec. 2000.

[26] M. Fowler. Using agile software process with offshore development. *martinfowler.com*, july 2004.

[27] R. Glass. Extreme programming: The good, the bad, and the bottom line. *IEEE Softw.*, 18(6):112, 2001.

[28] R. Grinter, J. Herbsleb, and D. Perry. The geography of coordination: dealing with distance in R&D work. In *GROUP'99*, pages 306–315. ACM, 1999.

[29] J. Herbsleb and R. Grinter. Splitting the organization and integrating the code: Conway's law revisited. *ICSE'99*, pages 85–95, 1999.

[30] J. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, June 2003.

[31] J. Herbsleb, A. Mockus, T. Finholt, and R. Grinter. Distance, dependencies, and delay in a global collaboration. In *CSCW'00*, pages 319–328. ACM, 2000.

[32] J. Herbsleb and D. Moitra. Global software development. *Software, IEEE*, 18(2):16–20, Mar/Apr 2001.

[33] J. Herbsleb, D. Paulish, and M. Bass. Global software development at siemens: experience from nine projects. In *ICSE'05*, pages 524–533. ACM, 2005.

[34] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE'07*, pages 188–198. IEEE Computer Society, 2007.

[35] J. Highsmith. Messy, exiting, and anxiety-ridden: Adaptive software development. *American Programmer*, 10(1), 1997.

[36] J. Highsmith. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Addison Wesley, 2000.

[37] J. Highsmith. *Agile Software Development Ecosystems*. Addison Wesley, 2002.

[38] B. Holmström Olsson, B. Fitzgerald, P. Ågerfalk, and E. Conchúir. Agile practices reduce distance in global software development. *Information Systems Development*, 23(3):7–18, 2006.

[39] H. Holmström Olsson, E. Conchúir, P. Ågerfalk, and B. Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. *ICGSE'06*, pages 3–11, Oct. 2006.

[40] L. Kiel. Experiences in distributed development: a case study. In *ICSE'03*, pages 44–47, 2003.

[41] M. Kircher, P. Jain, A. Corsaro, and D. Levine. Distributed extreme programming. In *XP'01*, pages 20–23, May 2001.

[42] A. Mockus and J. Herbsleb. Challenges of global software development. *METRICS'01*, pages 182–184, 2001.

[43] G. Olson and J. Olson. Distance matters. *Human-Computer Interaction*, 15(2/3):139–178., Sep 2000.

[44] M. Paasivaara and C. Lassenius. Using iterative and incremental processes in global software development. *IEE Seminar Digests*, 2004(912):42–47, 2004.

[45] M. Paasivaara and C. Lassenius. Could global software development benefit from agile methods? *ICGSE'06*, pages 109–113, Oct. 2006.

[46] S. Palmer and M. Felsing. *A Practical Guide to Feature-Driven Development*. Pearson Education, 2001.

[47] C. J. Poole. Distributed product development using extreme programming. In *XP'04*, volume 3092 of *Lecture Notes in Computer Science*, pages 60–67. Springer, 2004.

[48] R. Prikladnicki, J. Audy, D. Damian, and T. de Oliveira. Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring. *ICGSE'07*, pages 262–274, Aug. 2007.

[49] B. Ramesh, L. Cao, K. Mohan, and P. Xu. Can distributed software development be agile? *CACM*, 49(10):41–46, 2006.

[50] R. Sangwan, M. Bass, N. Mullick, D. Paulish, and J. Kazmeier. *Global Software Development Handbook*. Auerbach Publications, 2007.

[51] K. Schwaber. Scrum development process. In *OOPSLA'95*.

[52] K. Schwaber and B. M. *Agile Software Development with Scrum*. Alan R. Apt, first edition, 2001.

[53] M. Simons. Internationally agile. *InformIT*, march 2002.

[54] L. Williams. A survey of agile dev. methodologies. 2004.

[55] D. Zand. *The leadership triad : knowledge, trust, and power*. Oxford University Press, New York, 1997.