Becoming Agile in a Non-disruptive Way
Is It Possible?

Ilia Bider and Oscar Söderberg
DSV, Stockholm University, Stockholm, Sweden

Keywords: Agile, Software Development, Software Engineering, Challenges, Tacit Knowledge, Knowledge Transformation.

Abstract: Due to the increasing popularity of Agile Software Development (ASD), more software development teams are planning to transit to ASD. As ASD substantially differs from the traditional Software Development (TSD), there are a number of issues and challenges that needs to be overcome when transiting to ASD. One of the most difficult challenges here is acquiring an agile "mindset". The question arises whether it is possible to acquire this mindset with the minimum disruption of an already established TSD process. The paper tries to answer this question by developing a non-disruptive method of transition to ASD, while using a knowledge transformation perspective to identify the main features of ASD mindset and how it differs from the one of TSD. To map the current mindset and plan the movement to the mindset that is more agile, the paper suggests using a process modelling technique that considers the development process as a socio-technical system with components that correspond to the phases of the development process. The method suggested in the paper has been designed in connection to a business case of a development team interested to transit to agility in a non-disruptive manner.

1 INTRODUCTION

1.1 Formulating a Problem

Agile Software Development (ASD) has appeared as a reaction on the increasing rate of changes in system requirements, e.g. see (Highsmith et al., 2000): "requirements change at rates that swamp traditional methods". Since 15 years from its inception, ASD from a niched development methodology, mainly used in the web development, made its way to becoming one of the mainstream methodologies. This leads to organizations that use a phase-based methodology become more willing to move to ASD.

Due to the essential differences between the Traditional Software Development (TSD) and ASD, a transition from one to another is quite difficult and includes a number of challenges and pitfalls that are reported in research papers (Conboy et al., 2011; Hajjdiab and Taleb, 2011), books (Smith and Sidky, 2009), and practitioners blogs (Hunt, 2015). The main difficulty here is that an ASD team requires having a "mindset" that differs from the one of a TSD team.

There are a number of books, such as, (Hajjdiab and Taleb, 2011), that suggest methods for transiting from TSD to ASD. However, following these methods presumes that the decision to complete such a transition has been made, and risks attached to the transition understood. In addition, a decision on which brand of Agile, e.g. XP, or SCRUM, to try needs to be taken quite early in the transition process.

Understanding the transition risks and making a right for the given situation choice of the agile practice requires experience. Thus, such a transition has better chances for success if it is led by an experienced person, e.g. an agile coach. Even in this case, there is no guarantee of success. What is more, even if the transition was successful in the end, it could cause a disruption of the existing development process for quite long time. If the existing process does not work, taking the risk and introducing the disruption are fully justified. However, if the process works satisfactory, there could be doubts whether it make sense to jump into the unknown taking the risks and going through the disturbances without knowing whether a better development process will emerge after the transition has been completed.

In connection to the deliberations above, a question arises whether it is possible to gradually transit from TSD to ASD with the minimum

Bider, I. and Söderberg, O.
Becoming Agile in a Non-disruptive Way - Is It Possible?.
In *Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS 2016) - Volume 1*, pages 294-305
ISBN: 978-989-758-187-8

disruption of the existing development process? In other words, the question is whether there already exists a method of non-disruptive transition to ASD, and if not, whether such method can be devised. Ideally, such a method should improve the existing development process even before the full transition cycle has been completed. It should be also possible to delay taking the decision on which brand of ASD to use, and even stop the transition at some point being satisfied with what has been achieved, and not taking risks of going farther.

1.2 Overview of a Solution

This paper is a report on the research aimed at answering this question. To the best of our knowledge, there is no non-disruptive method of transition to ASD described in the research or practical literature. Therefore, we use Design Science (DS) approach (Peffers et al., 2007) to answer the question posed above, i.e. we aim to answer it by designing such a method and testing it in practice.

According to the case studies reported in the literature, e.g. (Hajjdiab and Taleb, 2011; Conboy et al., 2011), the biggest issue when transiting to ASD is acquiring the agile mindset by the development team. The latter requires all team to acquire a number of skills, which might not be necessary in the existing TSD. For example, social and communication skills are mandatory for all members, so that they can meet and talk to stakeholders. Therefore, the main focus of our design work is directed to acquiring the agile mindset and a set of skills that is included in it.

To design a method that leads to changing the mindset of the team to the agile mindset, we need to:

1. Find a basis on which to identify the main features of the agile mindset and in what way it differs from the mindset of a more traditional team.
2. Find a way of mapping (modelling) the mindset of the current team so that the difference between the current mindset and the targeted one (agile) can be measured and a plan of action aimed to shorten this distance can be developed.

As far as the first item on the list is concerned, the most commonly used framework for this kind of goal is Agile Manifesto (Agile Alliance, 2001). However, we consider it too vague and allowing multiple interpretations, which leads to misunderstandings and heated arguments in the agile community (Weaver, 2011); see also critique of Agile Manifesto in (Conboy and Fitzgerald, 2004). We needed a more "scientific" basis for developing a non-disruptive method of transition to agile. For this end, we have chosen an approach suggested in (Bider, 2014) that is based on considering TSD and ASD projects from the knowledge transformation perspective. Based on this consideration, (Bider, 2014) defines the essence of ASD in difference from TSD and set some requirements on the structure of the agile project, its team, relations with the customer and techniques used in the project. The results from (Bider, 2014) do not contradict Agile Manifesto, but rather more clearly underline the main features of ASD and the difference between ASD and TSD.

As far as the second item on the list above is concerned, there are a number of methods for evaluating and measuring the current level of agility, see for example (Sidky, 2007). However, mostly, these works rely on Agile Manifesto when determining what the agile mindset is. Furthermore, they are based on the decision of transition to agile being already taken. In addition, these are general methods not connected to the current structure of the development process accepted in the given organization. In other ways, we consider that the existing methods of evaluation of the level of agility do not fit the task of creating a method of non-disruptive transition to agile.

In this work, we have created our own approach to maping (modelling) the mindset of the development team that is suitable for planning steps for advancing the current mindset towards the agile one. This approach is based on the business process modelling technics suggested in (Bider and Perjons, 2015; Bider and Otto, 2015) and called step-relationship modelling in (Bider and Perjons, 2015). The technique uses a system view on the business process considering it as a number of components (or steps) connected with each other via various relationships. The model built according to this technique focuses on depicting these relationships and their properties. When adopting step-relationship modelling technique for our purpose, we concentrated on relationships between the teams that man the components/steps of the given system development process.

One of the main activities in a Design Science (DS) research project is testing the new artefact/solution, which is a method in our case, in at least one real situation. DS does not set a restriction on when in the course of the research project such test needs to be started, e.g. after the design has been finished or in parallel with the design. In our case, the research was conducted in parallel with investigating a business case in the IT department of an insurance company. This department was interested in adopting

a non-disruptive approach of moving towards agility, and it was also used as a test bed for the method. The test is far from being completed, but it was run up-to the department management understood enough of the suggested method and became prepared for completing the first step on the way to agility.

The rest of the paper is structured in the following manner. Section 2 gives a brief overview of the research methodology and knowledge base used in this research and the research background. Section 3 describes the proposed method. Section 4 discusses testing. Finally, in Section 5, we summarize the results achieved and draw plans for the future.

2 RESEARCH BACKGROUND

2.1 The Project History and Methodology

This research has been initiated by the management of an IT department in a large insurance company expressing their interest in transition to a more agile development process. The management did not possess much knowledge on the essence of ASD, or its various brands. They were interested in an approach that included minimum risks and gave a possibility to learn the essence of ASD on the way, while allowing to delay the decision of which particular brand/practice of ASD to adopt. The literary study, part of which is presented in Section 1, has shown that there are a number of practical methods of transition to agile. Nevertheless, none of them was particularly suitable for the requirements that came from the IT department. These requirements were reformulated into the question of "whether it is possible to gradually transit from TSD to ASD with the minimum disruption of the existing development process?" posed in Section 1.1 To answer this question, we decided to develop a "non-disruptive" method of transition to agile.

The development of our method follows the pattern of Design Science (DS) research (Peffers et al., 2007; Baskerville et al., 2009), which is related to finding new solutions for problems known or unknown. To count as a design science solution, it should be of a generic nature, i.e. applicable not only to one unique situation, but to a class of similar situations. DS research can be considered as an activity aimed at generating and testing hypotheses for future adoption by practice (Bider et al., 2013).

Our method development ran in parallel with the investigation of the business case of the IT department in the insurance company. More exactly, we investigated and modelled the structure of the development process in the department including the skill-sets of the process participants and the ways they communicated with each other. The activities were carried out through interviews with representatives of various phases in the process, and studying the internal documentation.

One of the key activity in a DS project is implementation and verification of a generic solution, or artefact in terms of (Peffers et al., 2007), in at least one situation. This activity is also referred to as demonstration or proof of concept in the literature devoted to methodology of DS (Peffers et al., 2007). The demonstration phase in this research is a continuation of our case study. More exactly, we worked out a suggestion on the first steps of the transition to agility for the IT department; and it was accepted by the management. More details on this activity are presented in Section 5.

As already has been mentioned in Section 1, we used some existing theoretical frameworks as a knowledge base when developing our method. As we do not expect that these frameworks are known to the reader, in the next sub-sections, we give a short overview of them before presenting our method.

2.2 Agility from the Knowledge Transformation Perspective

In this section, we give a short summary of TSD and ASD models built based on the knowledge transformation perspective presented in (Bider, 2014). These models, in their own turn, are built based on the SECI model (Nonaka, 1994). SECI stays for Socialization – Externalization – Combination – Internalization, and it explains the ways of how knowledge is created in an organization while being transformed from the tacit form (in the heads of the people) to the explicit one (e.g. on the paper) and back, see Figure 1. The cycle of knowledge creation consists of the following four steps or phases:

1. The cycle starts with *Socialization*, where tacit knowledge is transferred from the heads of one group of people to others via informal means, such as conversations during the coffee breaks, meetings, observations, working together, etc.
2. The next phase is *Externalization*, which is the conversion of knowledge from the tacit form into the explicit one, e.g. a model of situation.
3. The third phase is *Combination*, which is transforming the externalized (explicit) knowledge in a new form using existing knowledge, e.g. solution design principles.

4. The last phase is *Internalization*, which is converting the explicit knowledge, e.g. a solution, in the tacit knowledge of people who will apply this knowledge to any situation that warrants it.

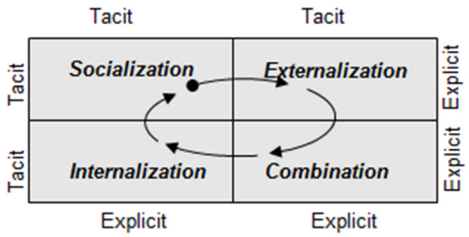

Figure 1: SECI diagram of knowledge creation.

Applying ideas from SECI to software development, (Bider, 2014) designed two models of knowledge transformation in software development projects, one - for Traditional Software Development (TSD), and another - for Agile Software Development (ASD). Both are presented in Figure 2. In both cases, the knowledge transformation cycles starts with tacit knowledge possessed by stakeholders on problems/needs to be solved/satisfied by a new software system. The next step common for both models is embedment when the knowledge on a solution becomes embedded in the system that is considered by its users as a whole possessing its own behaviour. The last step in the knowledge transformation in both models is adoption – transforming the knowledge embedded in the system into the tacit knowledge of the system's users on how to use this system in various working situations.

The models for TSD and ASD in Figure 2 substantially differ in the following aspects:

1. The nature of the first phase in ASD differs from that of TSD. It consists in transferring tacit knowledge on the problem and needs from the stakeholders to the development team. This phase corresponds to *Socialization* in Figure 2. Also, *Design* and *Coding* are merged into one phase Embedment. This can be defined as the first motto of agility: "Avoid or delay explication of knowledge as much as possible. Ideally go from tacit knowledge directly to the embedded one."
2. In addition, one big cycle is substituted by many smaller and shorter ones. The system is built iteratively starting with the basic functionality. During the exploitation of the basic system, better understanding of the needs is acquired, which is converted in adding details to the system in the next iterations. In other words, the second motto of agility can be defined as: "Develop and introduce in practice as little as possible as soon as possible, and build upon it in the following iterations".

Based on the analysis of the knowledge transformation models for TSD and ASD, (Bider, 2014) identifies 6 properties of the development process that differentiate TSD and ASD; these are presented in Table 1. The first three properties, *team*, *user involvement* and *agreement*, belong to the social

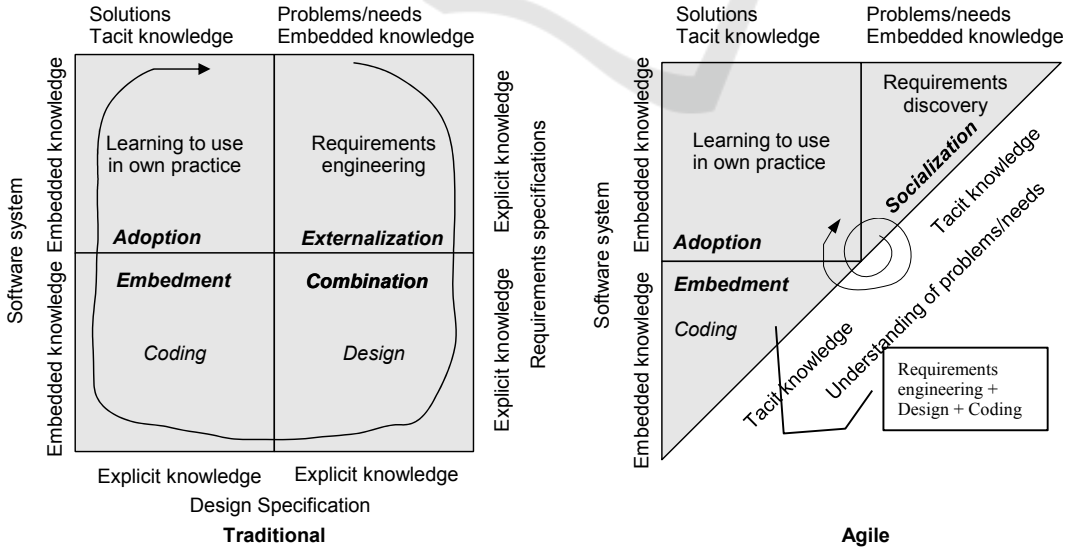

Figure 2: Left – ECEA model (Externalization-Combination-Embedment-Adoption) for TSD. Right - SEA model (Socialization-Embedment-Adoption) for ASD. Adapted from (Bider, 2014).

perspective of system development, while the second three properties, *core system*, *architecture* and *tools*, belong to the technical perspective of system development. We will be using these differentiating properties when developing our non-disruptive method later in Section 4.

Table 1: Properties that differentiate ASD from TSD.

#	ASD	TSD
1	One *team* consisting of "universal" members	Several specialized *teams*
2	*Stakeholders involvement* during the duration of the project	*Stakeholders involvement* during the *Externalization* and *Adoption* phases
3	Non-contractual *agreement* based on trust	*Contractual* agreement is possible
4	Possibility to identify and agree on a *core system* that can be expanded in consequent iterations	Not mandatory, but can be employed.
5	*Architecture* aimed at expansion	*Architecture* aimed at fulfilling the identified requirements
6	Employing *high-level tools*, e.g. domain-specific languages, development platforms, libraries	Not mandatory – low level, and universal tools can be employed

2.3 Step-relationship Model

A step-relationship model represents a business process as a (relatively) small number of steps (Bider and Perjons, 2015), or functional components (Bider and Otto, 2015), connected with each other through various types of relationships. Each type of relationships, i.e. a relation in a mathematical sense, represents a separate view of the model.

There are two ways of representing a relationships type, graphical and matrix. In the graphical form, the steps/components are presented as rectangles (boxes), while arrows between the rectangles show relationships between the corresponding steps/functional components. Labels inside the rectangles name the steps, while labels on the arrows give additional characteristics to the relationships. As an example, Figure 3 represents output-input relationships in a sample software development process. Each arrow shows formalized output of one step/component serving as an input to another step/component.

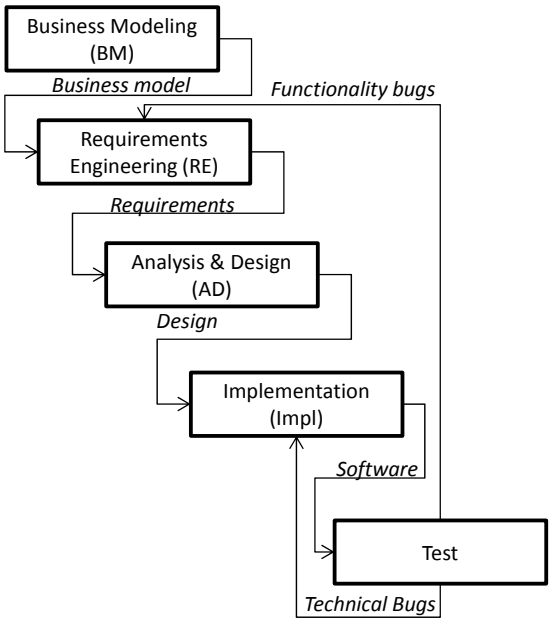

Figure 3: Graphical presentation of relationships.

In the matrix form, a relationships type is represented as a square matrix where both columns and rows correspond to steps/components of the process. A sell (*a*,*b*) where *a* is a column and *b* is a row is reserved for describing a relationship of the given type between step *a* and step *b*, if any exists. As an example, Table 2 presents the same output-input relationships type as Figure 3, but in the matrix form. More examples of relationships in the graphical and matrix forms are presented in Section 3.

Table 2: An example of presenting relationships in the matrix form.

	BM	RE	AD	Impl	Test
BM					
RE	Model				Bugs
AD		Reqs			
Impl			Design		Bugs
Test				Software	

3 DESIGNING A METHOD

3.1 Creating a Single Team

There are several essential properties of ASD that need to be achieved in order to successfully transit to agile. When developing our method, we assume that at least some of them can be achieved without essentially changing the current process. We also

assume that it is possible to somehow measure the progress achieved on the way.

According to the first row in Table 1, ASD has a single development team of members that could do all kind of work in the process, including talking to the stakeholders and programming. This is not mandatory for TSD, where separate specialized nonintersecting teams can complete the job. Also, in a single ASD team, all members communicate with each other frequently, which is not required in TSD. In TSD, informal communication in the frame of the development process may concentrate inside each specialized team, while the formal output-input channels are used for passing over the job between the teams, as is represented in Figure 3, and Table 2.

The two properties of (a) having specialized teams and (b) lack of communication between the teams are related to each other. A narrow specialization may create a hinder for communication due to differences in professional jargons and culture.

Based on the deliberation above, we have identified two properties of the development process that need to be measured and improved, in the first hand, when transiting to the agile approach. These are: (a) intensity of communication between the teams, and (b) ability of members of one specialized team to do the job assigned to the other teams. These two properties can be represented via relationships between the teams manning the steps. Technically, these relationships can be represented with the help of two matrixes: (a) the *communication intensity matrix*, and (b) the *cross-competency matrix*, as is discussed in the next subsections.

3.1.1 Increasing Communication Intensity

An example of the *communication intensity matrix* for the model in Figure 1 is presented in Table 3. A cell (*a,b*) in the communication intensity matrix, where *a* stays for a column and *b* for a row, defines the intensity of communication between teams of steps *a* and *b* initiated by team *a*. Interpretation of the values in the cells depends on the level of separation between the teams, e.g. one site or multiple sites. In the example presented in Table 3, communication are supposed to take place in the form of meetings, were *High* means daily communications meetings. *Average* means 3 times a week, *Low* means once a week. Empty cells outside the diagonal mean that no communication happens between the corresponding teams.

Note that the communication intensity matrix is aimed at characterizing the intensity of communication between the specialized teams, assumption being that inside the teams their members communicate/collaborate in a natural way. If this is not true, the diagonal of the matrix can be used for representing communication intensity inside the teams.

Table 3: An example of a communication intensity matrix.

	BM	RE	AD	Impl	Test
BM		High	Average		Low
RE	High		Average	High	Low
AD	High	High		High	
Impl	Low		Average		High
Test	Low	Low	Average	High	

The communication intensity matrix can be used for both depicting the communication intensity in the current state and planning for increasing the communication intensity. The latter can be done by changing values of some cells in the matrix to reflect the goal of increasing communication intensity. To facilitate the planning work, we have transferred some information from the output-input matrix, see Table 2, to the communication intensity matrix in Figure 3. More specifically, we make the borders of cell (*a,b*) thick in all cases where cell (*a,b*) is not empty in the output-input matrix. The latter means that the column step *a produces* a formalized input for the row step *b*, e.g. design specification. In addition, we made the background of cell (*a,b*) grey in case cell (*b,a*) is nonempty in the output-input matrix (Table 2). The later means that the column step *b receives* formalized output from the row step *b*.

Formally, the result of adding thick borders and grey background means that the matrix presented in Table 3 is a merger of a "pure" intensity communication matrix (without thick borders and grey background) with the simplified output/input matrix (the content of the cells in the latter is not represented in the merger) and a transposition of the latter. The merged communication extensity matrix is more convenient for planning the next step of transition to agile as described below.

One can expect that communication should be more extensive between the steps that are connected with an output-input relationship. Formalized outputs, like requirements or a design specification, in a software development process cannot be made totally formal, and they need interpretation from the receiving team. Misinterpretation can lead to a wrong system being delivered to the customer. The thick border represents the needs of informal explanation of the formalized output when it is being transferred to the receiving team. The grey background represents the need for communication between the

receiving team and the producing team while the former is doing their part of work. Even when the receiving team get the informal explanations on their formalized input, there can be a need to verify their understanding from the originator of the input. For example, the designers may need to contact the requirements engineers later on when they start converting certain requirements into design. In (Bider and Perjons, 2015), this type of backward communication is called week dependencies, while (Bider and Otto, 2015) refer to them as to feedback links.

Summarizing the above, when planning the next goal in intensifying the communication between the teams, it is worthwhile to start intensification that corresponds to cells with thick borders or grey background. For example, the next goal for the situation presented in Table 3, could be the one described in Table 4, where the difference is presented in bold. The difference consists of intensifying forward communication between *Analysis & Design* and *Implementation*, and backward communication between *Analysis & Design* and *Requirements Engineering*. Such measure makes sense even for improving the already existing process.

Table 4: Next step in communication intensity.

	BM	RE	AD	Impl	Test
BM		High	Average		Low
RE	High		**High**	High	Low
AD	High	High		High	
Impl	Low		**High**		High
Test	Low	Low	Average	High	

3.1.2 Increasing Cross-Competency

While the communication intensity matrix can be considered as a tool of intensifying internal communication in the future single team, the cross-competency matrix can be considered as a tool for achieving "universality" of its members (see the first row in Table 1). An example of such a matrix is presented in Table 5. In this matrix a cell (*a,b*), where *a* stays for a column and *b* for a row, defines the percentage of the team *a* members that have working knowledge on the tasks completed in the step *b*. An empty non-diagonal cell means 0%. Here, having working knowledge on a specific task means that a person in question has some practical experience of this task.

As with the communication intensity matrix, we add to this matrix some information from the output-input matrix in the form of thick borders around cells and grey background. This information is aimed at helping to plan the next step of transition to agile. Marked cells should be targeted for increasing cross-competence in the first place, as this can decrease the risks of misinterpretation of the formalized inputs and misunderstanding in communications. Such measure might be helpful even for improving the existing process.

Table 5: An example of cross-competency matrix.

	BM	RE	AD	Impl.	Test
BM		50%	75%		
RE	75%		75%		50%
AD	75%				
Impl.	50%	50%	75%		50%
Test	50%				

An example of the next planned step for the situation presented in Table 5 is presented in Table 6, where the difference is presented in bold. The difference consists of increasing cross-competency of the *Requirements Engineering* and *Implementation* Teams.

Table 6: next step in cross-competency.

	BM	RE	AD	Impl	Test
BM		50%	75%		
RE	75%		75%		50%
AD	75%	**50%**		**50%**	
Impl	50%	50%	75%		50%
Test	50%			**50%**	

As cross-competency requires *working* knowledge of the tasks completed by other teams, it is not enough just to send people to a course. The proper way of achieving cross-competency in cell (*a,b*) in the frame of the existing software development process is to send some people from team *a* to work in team *b* for some time. This can degrade the overall performance in the beginning, but this one-time cost is worth taking, as increase in cross-competency minimizes the risk of producing the wrong software (see deliberation above).

When planning increase in cross-competency for *Implementation* step with other teams, it is worthwhile to consider row 6 in Table 1 that refer to using high-level tools. This property has not been introduced for the sake of creating a single team of "universal" members, but for being able to complete development loops in a speedy manner. However, having high-level development tools may also help in acquiring programming skills by people without

technical education. So, if such tools are not already employed, it could be advantageous to start transition from low-level programming to using high-level tools before increasing competency in programming in other teams.

3.2 Avoiding Explication of Knowledge

As was discussed in Section 2.2, one of the ASD principles is to delay or avoid explication of knowledge, ideally, by going from the tacit understanding of problems/needs to building software. This implies skipping creating detailed requirements and design specifications. More specifically, requirements are left on the tacit level as a general understanding/image of the problems and needs, while design is done via proper structuring of the code. The latter could be facilitated by using high-level development tools, like domain specific languages, component libraries.

Avoiding explicit requirements and design does not mean that these activities are excluded; they are done on the tacit level. To reach the level of proficiency when requirements and design are done on the tacit level is difficult, if ever possible, without obtaining skills in both requirements engineering and design. Obtaining these skills by all team members in the frame of the existing phase-based process has already been discussed in Section 3.1.

The next question is how to shorten the time period from the first contact with the customer to starting producing executable code while still remaining in the frame of a traditional software development project. We believe that this can be achieved by gradual transition from sequential execution of the steps of development process to the semi-parallel execution. The latter means starting the design before all requirements are discovered, and starting coding before all design specifications are created.

The current level of parallelism can be represented in a graphical form as a timeline intensity diagram (Bider and Otto, 2015). An example of such a diagram that corresponds to Figure 3 is presented in Figure 4. The difference between Figure 3 and 4 is that in Figure 4, the shapes representing steps do not have rectangular form. The upper border of the shape can be of any form representing the increase/decrease in the amount of work being done at certain moments of time. The intensity of work can be increasing or decreasing with time, or can be first increasing and then decreasing or vice versa (not illustrated in Figure 4). In addition, the step shapes in Figure 4 are placed in the order they are executed. If some steps run partly in parallel, the projections of their shapes on the time axes will intersect. In the example of Figure 4, there are two occasions of the parallelism, namely (1) step *Analysis & Design* runs partly in parallel with *Implementation*, and step *Implementation* runs partly in parallel with *Test*.

Timeline intensity diagram can be used for planning the next goal for transition to agile in the same way as communication intensity and cross-competency matrices are used, see Fig. 5.

In the example of Figure 5, all steps run partially in parallel, which is rather a radical change when starting from Figure 4. If such a transition is too difficult to complete in one go, then smaller goals can

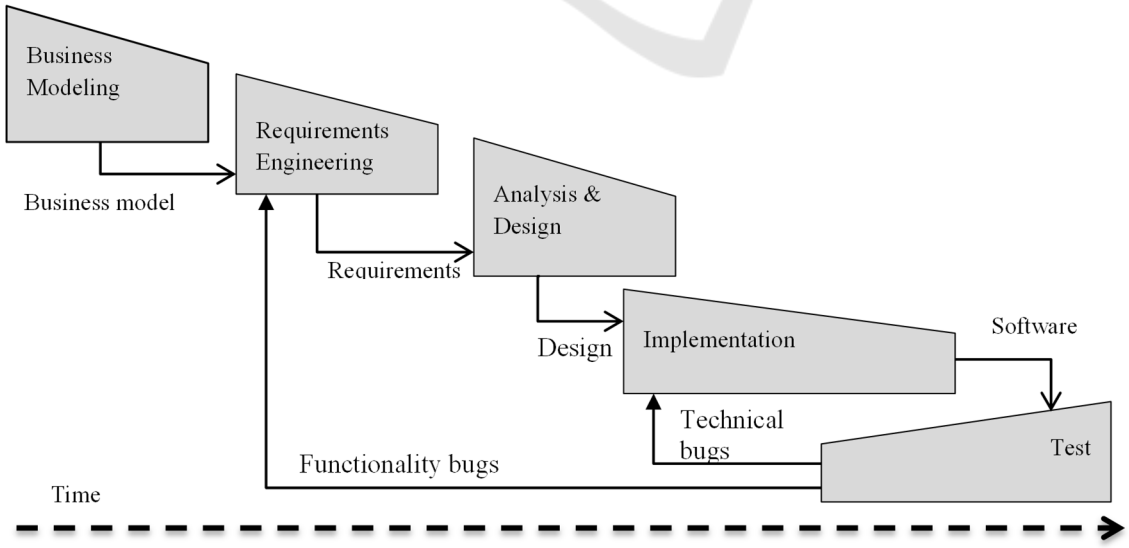

Figure 4: An example of timeline intensity diagram.

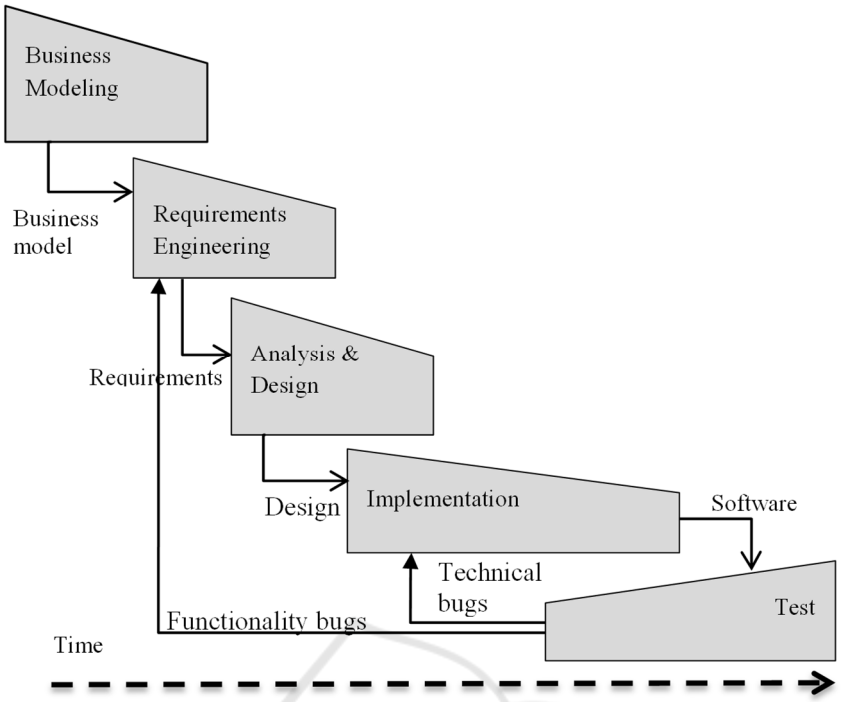

Figure 5: An example of timeline intensity diagram to be achieved.

be set in between, e.g. where only two new steps run in parallel.

Working in parallel means that the formalized output is delivered to the next step in portions. This requires understanding of how the formalized output is used by the next step so that each portion is relatively independent and can be successfully used by the team of the next step for producing its own formalized output. Thus parallel execution requires certain degree of cross-competency on behave of the output producer. In addition, it requires efficient communication channels between the steps. Parallel execution of steps in software development bares a risk that the already produced portion of the given step output, e.g. requirements, can be negated when the work progresses. If this "negated" portion has already been sent to the next step, e.g. design, and is under processing of this step's team, then the information on the negation should be immediately made available for this team. Getting this information can stop or postpone their activities related to the questionable portion of the requirements. Note that with an experienced team, the advantages of running in parallel, e.g. shorten time, overweight the risks described above.

Summarizing the deliberation above, transition to parallel execution of two steps should be planned when a certain degree of cross-competency and communication intensity between these steps has already been achieved.

It is also worthwhile to mention that portioning of the output needs to take into account architectural considerations. Portions that are sent first need to be significant for building a skeleton of the architecture, and portions that are sent later should be relatively independent of each other and should not considerably affect the architecture.

3.3 Other Considerations

In the previous part of this section we mainly discussed three issues that can help in transition to agile: inter- step communication, cross-competency and parallel execution. Furthermore, we touched the issue of high-level development tools that facilitates both achieving cross-competency, and excluding explicit design. In addition, we also touched the architectural issues that need to be taken care of when planning transition to the parallel execution of steps. We also have shown that all these issues are interconnected and should be considered together when planning transition to agile.

We believe that after dealing with the issues discussed in this section the team will acquire the agile mindset, and became prepared for sorting out the remaining issue on the way to agility. Consider, for example, the issue of stakeholders involvement during the whole project. Such involvement is

impossible to arrange in a traditional phase based development process based on two reasons. Firstly, people outside business modelling and requirements engineers might not have competency of talking to non-technical people. Secondly, non-technical stakeholders seldom understand technical documentation, which will prevent their engagement. The first problem can be solved through cross-competency, and the second - through parallel execution that ensures that the new portion of software will be produced in a speedy fashion, and could be demonstrated and discussed with the stakeholders.

4 TESTING THE METHOD

As has already been discussed in Section 2.1, development of our non-disruptive method of transition to agile was done in parallel with a case study in the IT department of a large insurance company. The first phase of the study was connected to the development of the method, and the second phase with testing it.

The first phase was completed based on the internal process documentation and interviews with representatives of different teams engaged in the development process. Based on the information obtained, it was decided that the three most important aspects that need to be mapped when describing the current state of affairs were communication intensity, cross-competency and timeline intensity. The step relationship modelling technique (Bider and Perjons, 2015; Bider and Otto, 2015) was chosen for representing these aspects. The concept of the timeline diagram was already known from (Bider and Otto, 2015), while the communication intensity matrix and cross-competency matrix where designed during the current project.

Based on the internal documentation and information from the interviews, a model of the current development process was produced. This model is closed to the one presented in Figure 3 and 4, and Tables 2, 3 and 5, except that one step from the original model is omitted. The structure of the communication intensity and cross-competency matrixes in the original model were somewhat simpler than what was presented in Tables 3 and 5. More exactly, the details that came from merging with the output-input matrix were absent; they were added when we worked on this paper.

The test phase of the case study consisted of: (a) suggesting the next desired state of the development project, which roughly corresponds to the one presented in Tables 4 and 5 and Figure 5, and (b) presenting the suggestions to the IT department management. The goal of the test phase was twofold, namely, to check

1. Whether the method could be understood by people not very familiar with the agile practices.
2. Whether they can accept concrete suggestions based on this method, provided that they are approved by the higher management. This check (approximately) corresponds to "reediness to use" in Technology Acceptance Model (Davis, 1989).

The check has been completed by presenting the method and an action plan based on this method to the management of IT-department that consisted of 4 persons. After the presentation, an interview has been conducted with each person based on the following 4 questions/topics:

1. Based on the presentation, have you understood what kind of organizational changes the transition to agile will require?
2. Based on the presentation, have you understood the action plan for movement towards a more agile development process?
3. Based on the presentation, are you prepared to submit the action plan to the upper/higher management for approval?
4. Based on the presentation, are you prepared to set the suggested plan in action if approved by the higher management?

For the questions 1, 2 and 4 the answers were on the positive side from all respondents. When answering question 3, some respondents expressed doubts whether just presenting the action plan to the higher management is enough to influence the approval. However, all of them agreed that such a presentation makes sense. The doubts on influencing the decisions were connected to the plan itself not explaining the benefits to be obtained. However, another opinion was that presenting the action plan could initiate discussions that would lead to understanding the benefits. Anyway, the discussion around the third topic explicated the needs to explain the benefits achieved even before the full transition to agile has been completed. This served us as a motivation to insert the discussion on such benefits in various places of this paper.

Summarizing the lessons learned about our non-disruptive method of transition to agile from the case study, we can state that:

1. It is possible to model the current state of the

development process and suggest a plan of actions for transition to agile.

2. The method is understandable for the professionals in software development not familiar with the details of the agile practices. What is more, the plan of actions based on the method is considered to be "doable", and could be accepted for implementation, provided the approval of the higher management is obtained. Though, there are some doubts that such approval is easy to obtain, presenting the plan of action to the higher management could initiate a discussion that could lead to its acceptance.

The lessons above were obtained based only on one case study. However, from our practical experience, the IT department in the study is just an ordinary system development organization, and there is no reason to suggest that the lessons learned will substantially differ when the method is applied to another organization of the same kind.

In short, we consider the check for "readiness to use" as completed with positive results. On its own, such a check does not guarantee that an organization can actually execute a plan of action developed based on the method. However, we consider this check encouraging enough for continuing the efforts of further development and testing the method.

5 CONCLUSIONS

There are ample evidence, provided in the literature referenced to in Section 1, of existence of challenges and difficulties when completing a transition from TSD to ASD. These can be attributed to such a transition being a major organizational change for a software development organization, and it is well known that any organizational change is difficult to complete due to an organization, as a system, always resists any change.

According to (Regev, 2015), the best prerequisite for successful organizational change is stability. Therefore, a system development organization with a well-functioning TSD process does not need to "jump" on a radical pass to ASD, but should consider using the existing process as a tool for successful transition to ASD. The non-disruptive method of transition to ASD described in this paper gives an example, of how an organization can practically conduct the transition via using the existing process as a tool.

Summarizing the results achieved so far, we can identify three major contribution of this work:

1. To the best of our knowledge, the contemporary literature does not have an explicit definition of a goal of using the existing development process as a platform/tool for transiting to agile. Therefore, our explicit formulation of this goal constitutes the first contribution of this paper. This contribution appears in the title and is discussed in more details in Section 1.
2. In Section 3, we have introduced three types of measurements that can be used to determine the level of agility achieved while the organization is still following TSD: communication intensity, cross-competency, and the level of parallelism. This are easy to understand measures, and as our test case shows can be obtain through interviewing people working in the project. This measures can be used independently whether the organization wants to transit to agile in disruptive or non-disruptive manner.
3. Lastly, this paper also contains a draft of the non-disruptive method of transition to agile that has gone through the initial test of designing a plan of actions and acquiring "readiness to use" in a typical software organization. More tests and further development are required to confirm the validity of the method. However, the work done so far (including the initial test) is sufficient to show that, at least theoretically, a non-disruptive method for transition to agility can be built. Publishing this work might inspire other researchers and practitioners to seek own ways for a non-disruptive transition.

One difference of our method of transition to agile with those of other (some of them are referred to in Section 1) is that we pursue a special goal of using the current development process as a tool/platform for the transition. Another difference is the theoretical basis on which the method has been built. Normally, other researchers and practitioners use Agile Manifesto (Agile Alliance, 2001) as a basis for building a method. Instead, we use the theoretical underpinning of agility based on the knowledge transformation perspective from (Bider, 2014). This perspective has helped us to choose the most important issues on which to focus when transiting to agile. What is more, the issues, when resolved, may improve the current development process even before the full transition can be completed.

Our plans for the future include further development and testing of the non-disruptive method, as well as dissemination of results, especially

among practitioners. The latter activity is considered as an important one in the Design Science research (Peffers et al., 2007). The reason for its importance is that the researchers themselves have no possibility to fully test a new design, aside of conducting demonstration in few cases. The real test can be completed only when (and if) the industry adopts the method so that more test cases become available for study.

ACKNOWLEDGEMENTS

The authors are in debts to the management and developers of the IT department of the insurance company who initiated this research and spent their time answering interview questions, and listening, discussing and accepting our suggestions.

REFERENCES

Agile Alliance, 2001. *Manifesto for Agile Software Development*. (Online) Available at: http:// agilemanifesto.org (Accessed 10 October 2013).

Baskerville, R.L., Pries-Heje, J. & Venable, J., 2009. Soft Design Science Methodology. In *DERIST 2009*. ACM, pp.1-11.

Bider, I., 2014. Analysis of Agile Software Development from the Knowledge Transformation Perspective. In Johansson, B., ed. *13th International Conference on Perspectives in Business Informatics Research (BIR 2014)*. Lund, Sweden. Springer, LNBIP 194, pp.143-57.

Bider, I., Johannesson, P. & Perjons, E., 2013. Design science research as movement between individual and generic situation-problem-solution spaces. In Baskerville, R., De Marco, M. & and Spagnoletti, P. *Organizational Systems. An Interdisciplinary Discourse*. Springer. pp.35-61.

Bider, I. & Otto, H., 2015. Modeling a Global Software Development Project as a Complex Socio-Technical System to Facilitate Risk Management and Improve the Project Structure. In *Proceedings of the 10th IEEE International Conference on Global Software Engineering (ICGSE), forthcoming*. Ciudad Real, Spain. IEEE.

Bider, I. & Perjons, E., 2015. Design science in action: developing a modeling technique for eliciting requirements on business process management (BPM) tools. *Software & Systems Modeling*, 14(3), pp.1159-88.

Conboy, K., Coyle, S., Wang, X. & Pikkarainen, M., 2011. People over Process: Key Challenges in Agile Development. *IEEE Software*, 28(4), pp.48-57.

Conboy, K. & Fitzgerald, B., 2004. Toward a conceptual framework of agile methods: a study of agility in different disciplines. In *Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*. Newport Beach. ACM, pp.37-44.

Davis, F.D., 1989. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3), pp.319–40.

Hajjdiab, H. & Taleb, A., 2011. Adopting Agile Software Development: Issues and Challenges. *IJMVSC*, 2(3), pp.1-10.

Highsmith, J., Orr, K. & Cockburn, A., 2000. *E-Business Application Delivery, pp. 4-17*. (Online) Available at: www.cutter.com/freestuff/ead0002.pdf .

Hunt, A., 2015. *The Failure of Agile*. (Online) Available at: http://blog.toolshed.com/2015/05/the-failure-of-agile.html (Accessed October 2015).

Nonaka, I., 1994. A dynamic theory of organizational knowledge creation. *Organ. Sci.*, 5(1), pp.14–37.

Peffers, K., Tuunanen, T., Rothenberger, M.A. & Chatterjee, S., 2007. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3), pp.45-78.

Regev, G., 2015. *Fundamental Systems Thinking Concepts for IS Engineering: Balancing between Change and Non-change*. (Online) Stockholm University Available at: http://sched.co/2OGV (Accessed October 2015).

Sidky, A., 2007. *A structured Approach to Adopting Agile Practices: The Agile Adoption Framework. PhD Thesis*. (Online) VirginiaTech Available at: http://scholar.lib.vt.edu/theses/available/etd-05252007-110748/ (Accessed October 2015).

Smith, G. & Sidky, A., 2009. *Becoming Agile*. Greenwich, CT: Manning.

Weaver, M., 2011. *Do you agree or disagree that Scrum is not Agile?* (Online) Available at: http:// www.linkedin.com/groups/Do-you-agree-disagree-that-81780.S.52354777 (Accessed April 2014).