Agile Experiences in Software Development

D. Duka
Ericsson Nikola Tesla, Split, Croatia
e-mail: denis.duka@ericsson.com

Abstract - From the traditional waterfall model to streamlined ways of software development and to Agile, is a long journey for each organization. In that journey, transition into Agile way of working means a significant change. For any change to happen there is a significant involvement expected from the people, more so in case of Agile, as it involves individuals, interactions and collaborations over different processes. This paper deals with different Agile experiences. It proposes when to use Agile and what are the advantages of such approach. It also describes agility criteria helping to determine if the Agile is followed indeed. The special highlight is put on people and teams investigating their size and impact of geographical distribution on project success. Some reflections after using Agile in our own organization are also presented.

I. INTRODUCTION

Agile software development is an evolutionary, highly collaborative, disciplined, quality-focused approach to software development and delivery, whereby potentially shippable working software is produced at regular intervals for review and course correction [1].

Agile software development techniques have taken the industry by storm, with 76% of organizations reporting in 2009 that they had adopted Agile techniques, and that on average 44% of the project teams within those organizations had adopted one or more techniques [2]. Agile development is becoming widespread because it works well – organizations are finding that Agile and iterative project teams, when compared to traditional project teams, enjoy higher success rates, deliver higher quality, have greater levels of stakeholder satisfaction, provide better return on investment (ROI), and deliver systems to market sooner [3]. By following quality techniques such as refactoring and developer regression testing throughout the lifecycle, agilists are able to progress safely and surely, increasing their productivity. By working closely with stakeholders in an iterative manner they have a better understanding of what stakeholders actually need and are more likely to deliver solutions that people actually want to use for their business purposes. By working in priority order, Agile teams are able to provide the greatest return on investment as defined by their stakeholders. In short, Agile teams work smarter, not harder, and thereby achieve better results.

Agile was once considered viable only for small, co-located teams. More recently, improvements in product quality, team efficiency and on-time delivery, all attributable to Agile practices, have caused larger teams to take a closer look at adopting Agile principles in their environments. A recent study conducted by the Agile Journal [4] determined that 88% of companies, many with over 10,000 employees, are using or evaluating Agile practices on their projects. Agile is truly poised to become the dominant software development paradigm. This trend is also echoed in other industry studies, including one conducted by Dr. Dobb’s Journal [5] which found a 76% adoption rate of Agile techniques, and within those organizations doing Agile, 44% of the project teams on average are apply in Agile techniques in some way.

Unfortunately, we need to take adoption rate survey results with a grain of salt: A subsequent another survey found that only 53% of people claiming to be on “Agile teams” actually were [6]. It is clear that Agile methods have been overly hyped by various media over the years, leading to abuse and misuse; in fact, the received message regarding Agile appears to have justified using little or no process at all. For too many project teams this resulted in anarchy and chaos, leading to project failures and a backlash from the IT community that prefers more traditional approaches. Properly executed, Agile is not an excuse to be undisciplined. It is clear that the execution of mainstream Agile methods have always demanded a disciplined approach, certainly more than traditional approaches such as waterfall.

It should not be surprising that working together closely in co-located teams and collaborating in a unified manner towards a goal of producing working software produces results superior to those based on working in specialized silos concerned with individual rather than team performance. It should also come as no surprise that reducing documentation and administrative bureaucracy saves money and speeds up delivery.

However, why wouldn’t we always use Agile methods? Well, maybe they are not always the best approach for the circumstances! So how do we decide? It is tempting to thinking that all projects should be handled in an Agile way.

Indeed, projects would benefit from the improved collaboration and communications encouraged on Agile projects. However, collaboration and communications are just two attributes of Agile projects and we must consider wider parameters that influence project success or failure.
The **Slider** (presented in Figure 1) is a simple model that illustrates that the Agile/Traditional choice is not necessarily binary [7].

![Figure 1. The Slider model](image1)

The model is fashioned after an old fashioned car heater control mixer, allowing for “No”, a “Mixture”, or a “fully Agile” approach as indicated by the position of the arrow. Characteristics that would pull the pointer one way or another are depicted as weights. Some project characteristics are a good fit Agile and would pull the pointer towards the Agile scale. Other characteristics make Agile introduction more problematic and might instead favor a more traditional approach. Traditional and Agile approaches can also be successfully combined to effectively address hybrid or hard to classify projects.

This paper deals with implementing the Agile approach in software development. It starts with exploring the reasons behind moving to Agile approach which is given in the following chapter. The paper continues with determining the team agility criteria trying to underline the difference between *doing* Agile and *being* Agile. Size of teams, their distribution and impact of geographical distribution on project success in Agile are presented in the fourth chapter. The final chapter deals with real experiences following the Agile implementation in R&D section in Ericsson highlighting some changes noticed during this new way of working.

**II. WHY AGILE SW DEVELOPMENT**

One of several reasons why Agile techniques are so effective is that they reduce the feedback cycle between the generation of an idea (perhaps a requirement or a design strategy) and the realization of that idea. This not only minimizes the risk of misunderstanding, it also reduces the cost of addressing any mistakes.

Figure 2 depicts the traditional cost of change curve, which shows that the longer it takes you to find a defect then on average the more expensive it is to address. The average cost rises exponentially the longer that you wait because you continue to build upon a shaky foundation. To be fair, some defects become less expensive the longer it takes to find them, perhaps the defect is found in functionality that is no longer needed or perhaps the defect is now addressable by a reusable component that is inexpensive to deploy [8].

![Figure 2. The traditional cost of change curve](image2)

In the past the X-axis of this curve was depicted in terms of the traditional project phases (requirements, analysis, architecture, design, ...) but the real issue really is one of the length of the feedback cycle. Although the cost of change curve has been questioned since it was first proposed by Barry Boehm in *Software Engineering Economics* (Prentice Hall, 1981), Boehm had looked at overly bureaucratic environments (mostly US Government and contractors to them), the real issue seems to be around how steep the curve is. As an aside, modern development processes have abandoned the idea of phases in favor of an evolutionary (iterative and incremental) if not Agile (evolutionary and highly collaborative) approach.

Figure 3 maps the feedback cycle of common development techniques to the cost curve. Agile approaches, such as Test Driven Design (TDD), Model Storming (MS), Agile Model Driven Development (AMDD) etc., all have very short feedback cycles, often on the order of minutes or hours. Traditional approaches, such as reviews, inspections, and Big Requirements Up Front (BRUF) have feedback cycles on the order of weeks or months, making them riskier and expensive [8].

![Figure 3. Mapping common techniques to the cost curve](image3)
Figure 4 presents a cost of change curve that can be expected for Agile software development projects.

As can be seen, the curve doesn’t completely flatten but in fact rises gently over time. There are several reasons for this:

- Minimally business code and test code bases will grow over time, increasing the chance that any change that does occur will touch more things later in the project.
- Not only code base increase over time. There will be documents such as user manuals, operations manuals and system overview documents that you will need to update. There are models, perhaps a requirements or architectural models, that will also be updated over time. Taking an AMDD approach will help to reduce the cost of this but will not fully eliminate it.
- When it becomes expensive to release software, perhaps you choose to distribute using some media instead of releasing software electronically to shared servers. Consequently, your cost of change increases because you begin to follow more conservative and more expensive procedures.
- Many Agile software development teams find themselves in very non-Agile environments and as a result are forced to follow procedures, such as additional paperwork or technical reviews, that increase their overall costs. These procedures not only increase the feedback loop but they are also very often not conducive to supporting change.
- Agilists create high-value documentation, or in other words, far less documentation than traditionalists do, and don't tolerate unnecessary bureaucracy. As a result they can act on changes quicker because they have less work to do. In other words, they maximize the amount of work not done.

### III. CRITERIA TO DETERMINE IF A TEAM IS AGILE

A common problem in many organizations is that undisciplined ad-hoc teams often claim to be Agile, because they’ve read an article or two about Agile development and interpret agility to mean any cool, libered form of undocumented software creativity. These ad-hoc teams often run into trouble, and give actual Agile teams a bad name. The following five criteria determine if a team is truly Agile [1]:

- **Working software** - Agile teams produce working software on a regular basis, typically in the context of short, stable, time-boxed iterations.
- **Active stakeholder participation** - Agile teams work closely with their stakeholders, ideally on a daily basis.
- **Regression testing** - Agile teams do, at a minimum, continuous developer regression testing. Some Agile teams take a TDD approach.
- **Organization** - Agile teams are self-organizing and disciplined teams working within an appropriate governance framework at a sustainable pace. Agile teams are also cross-functional “whole teams,” with enough people with the appropriate skills to address the goals of the team.
- **Improvement** - Agile teams are disciplined and also measure, how they work together and then act to improve on their findings in a timely manner. Continuous improvement is a key aspect for success in all organizations using Agile.

An important aspect of these criteria is that they are flexible. If we analyze the terms used in the description of the criteria – regular basis, closely, continuous, appropriately, regularly, timely; they are all situational in nature. For example, for some teams regular basis might be once every week, for other teams in more complex situations once every six weeks. Because every team finds itself in a unique situation, they must be flexible in the way that they assess their agility. The real goal is to be as effective as possible given the situation.

In fact, it is quite easy to do Agile. We pick the non-threatening pieces and parts and simply do those. Unfortunately, in doesn't mean we are doing Agile very well.

*Being Agile is completely different. Being Agile means we understand the principles which lead to true Agile success. It also means the team and organization are both constantly improving. When we are being Agile, daily standup meetings and retrospectives are both very important meetings which help the team to be successful. Finally, being Agile means being unafraid of failure. Doing Agile has none of these qualities because it is all about doing the Agile practices, not living the Agile principles!*
What is the right way from doing Agile to being Agile? One possible approach is understanding the difference. Once we understand the difference we can start to rely on the process to self-correct you toward being Agile.

For example, retrospective meetings might be a right place for new proposals based on different questions e.g. do we need a new way of communication flow, why do we not meet the commitment made during iteration planning, is the team willing to be Agile and work toward continuous improvement rather than continuous mediocrity, etc.

Once the questions are asked the team should strive to find some action items which will help them get better in those areas. If the team is truly dedicated to being Agile rather than doing Agile they will find action items which they can commit to in order to improve. This is the key first step to being Agile. Once we have a breakthrough in this area it is very easy to continue being more Agile [9].

IV. AGILE AND TEAM STRUCTURE

An interesting survey was performed in July 2009. The goal was to investigate how size of the teams and their geographical distribution influence the project success.

Figure 5 summarizes results from the Agile Practices Survey [10] which asked about Agile team size. Although the majority of Agile teams are 20 people or less, some organizations do in fact have fairly large Agile teams. Team size increases communication and organizational risks on Agile delivery teams and is often an indication of both technical and organizational complexity (the more complex the situation, often the larger the team required to address that situation).

Figure 6 summarizes results of a question from the same survey asking about Agile and geographical distribution. Apparently, only a minority of Agile teams are co-located, with the rest exhibiting some form of geographical distribution.

Figure 7 summarizes the results of several questions from the how the project success rate decreases as your team becomes more geographically distributed (Near Located in Figure 7 is the combination of Same Building and Within Driving Distance in Figure 6). It is interesting to note that some organizations are in fact succeeding even when their Agile teams are very distributed.

In order to have full understanding of figure above it is critical to understand how people actually define process success.

- Functionality: 83% believe that meeting actual needs of stakeholders is more important than building the system to specification.
- Quality: 82% believe that delivering high quality is more important than delivering on time and on budget.
- Money: 70% believe that providing the best ROI is more important than delivering under budget.
- Schedule: 58% believe that delivering when the system is ready to be shipped is more important than delivering on schedule.

Nevertheless, as a conclusion, geographical distribution increases the communication risks faced by Agile delivery team, which in turn decreases your success rate, requiring you to adopt your process accordingly to hopefully address those risks.
V. AGILE EXPERIENCES

Agile approach was implemented in one R&D section at Ericsson. The section was chosen since it was targeted to take responsibility for approaching international project matching following criteria: new development including pre-study/feasibility activities, combining both design and test activities, active quality and tool support. Experiences after using Agile as well as some changes are highlighted below.

When using Agile approach in software development, everything begins with a Product Backlog. This product backlog, just like any other product backlog, has high-level requirements and the expected timelines. Every item in the product backlog had a clear Definition-of-Done (DoD), prioritization and was sized and estimated [11].

The coach role is very important. The coaches, who are none other than the engineers from the development community, spread the message of good Agile practices into the teams, and making the teams see the benefits through their own eyes. Coaching happened in various forms: motivational, consultative and educational. Initially, the teams were in an educational mode, trying to understand what Agile is, what the practices are. The basic Agile workshops were given, for the various Agile roles and for team members.

Once the basic understanding was there, there were more queries on how and why. The coach’s role here was more consultative in nature. The teams actively took help from them to understand how to practice Agile in a proper form. For example, the teams started estimating, doing planning and other key aspects well. The coaches had a high level view of how the teams were practicing Agile and helped them share good practices across teams. There were many motivational aspects that needed to be supported as well.

Sprint planning is crucial. The User Stories from the prioritized backlog were picked up by the teams across the three sites. It was always “Think Global and Act Local”. So all common strategies were laid down at a global level and the implementation of them was taken care at the local level, with a common vision.

The heartbeat of the team was in stand-up meetings. There was also a monthly sync-up across all three sites. In this sync-up we updated each other on what was done, what are good practices, what are the impediments that teams faced, etc.

A. Self-organizing Teams

One of the critical ingredients for success of Agile are self-organizing Agile Teams. Traditionally, the teams had been working on detailed requirements being provided by the parent organization. They were used to being told what to do and even how to do. Self-organizing requires the teams to be willing to go the extra mile in identifying what needs to be done, how and when it needs to be done. So the team needs to demonstrate a lot more ownership on their tasks, taking them to a closure by themselves.

Ownership is a lot to do with self-belief; with each individual seeing their own potential and trusting themselves to do better. While the Line Managers stood by their people, any failures were seen as a step towards success, encouraging them to keep trying, and doing better than the previous attempts.

Periodic retrospectives (outside of the retrospectives at the end of iterations) helped to focus initially not on the tasks of the completed iteration, but on the Agile practices, the ownership behaviors that were demonstrated besides other softer aspects. These retrospectives helped the team to do a lot of introspection and realize their own potential and set the bar higher each time. When the team members saw that their managers are not watching them to find fault, but to hold them from falling down, they grew in confidence. The culmination of this effort was, when the teams took on the ownership for these retrospectives and the managers started to step back. The team reported their retrospective findings, and the actions that they, as a team were committing themselves to. The creation of the self-organizing teams had begun and in no time was gathering a lot of momentum.

The Line Managers gave up a lot of control and took a step back, when dealing with self-organizing teams. Once the self-organized teams were in place, the decision-making process underwent considerable changes with the teams feeling the need to be involved. Consequently, a more shared ownership has evolved in decision making since teams were actively involved in giving their views and recommendations to the Line Managers.

In some of the teams, the team members resorted to pair-programming without being asked to try it out, improving quality and reducing the time on inspections and handovers. This became a point for learning for the other teams.

Change happened: from high management commitment to high worker commitment, from management controlled to self controlled.

B. Cross-functional Teams

Every activity has different tasks attached to it, requiring different skills. Agile teams were created in a manner that it was balanced with regard to the skills required. Testing was also included in every team. When dealing with teams that are about 5-6 members having dedicated testers was not a preferred option compared to having a domain or technology expert. In order to change this situation, testers’ role were played in rotation; which ever specific skills were not going to be much used in a particular iteration, the related engineers took on the role of the tester for that iteration. This ability to switch between development and testing proved to be very useful in optimizing the size and the skills available in a team.
Change happened: from isolated specialists to multi-skilled workforce.

C. Stable Teams

Most significant aspect of Agile is prioritizing people and interactions over documentations and processes. When we talk about people and interactions, a lot is to be gained through the relationship and rapport that is built amongst team members. To enable this relationship to be built resulting in intra-team trust, it is essential to keep the members of the teams stable. A team structure was put up and consciously kept in place for long periods. Any changes that had to be done, for example on account of attrition of a member of a team, would result in replacement being fitted into that same vacancy in that team, thereby not disturbing any other team.

To keep stable teams, would mean that every team should be capable enough to handle any of the requirements of the project. Otherwise, there would be situations where one team is loaded with a lot of things to do, while another does not have enough on its plate. The creation of the teams to be cross-functional came in handy here. As a result, stability of the Agile teams over long periods of time could be achieved.

Stability in team also brought in another benefit—a predictable pace of work, helping in better and more predictable schedules and commitments from a project release perspective. Also over a period of time, the team picks up a lot of self-discipline aspects which are key to success.

Change happened: from a management driven team to self-organized team.

VI. CONCLUSION

Agile approach begins with an Agile belief with all individuals in the organization. It has to be the entire team knowing how to play and move towards a common goal. Similarly, every team member needs to understand the value of Agile and how it has to be played/practiced and be Agile while developing the software. Successfully implementing organizational wide change requires extra time and high energy level from the whole organization.

Our motivation behind the change was not only the current product under development but also to increase the level of Ericsson R&D center ability. First-hand experience of Agile framework is the real source of learning. Preliminary evidence we’ve collected from a team survey, rework, velocity, point growth, acceptance test defect trends, and other sources all tell us that Agile software development is making a positive difference in our organization. In addition, interest and respect for Agile practices is growing. Other software teams are adopting Agile practices and requests for our guidance and support are increasing. Other parts of the company are interested in what we’re doing and how we’re making it work. It’s required our resolve, our persistence and some measurable success, but Agile practices are taking hold in our culture.

To summarize, by using this approach in software design we are able to first recognize and then eliminate activities which don’t add value e.g. partially done work without any guarantee if customer will take it in use, unnecessary task swapping, waiting, handoffs, faults, etc. This way of working also implies improved responsiveness through rapid deliveries allowing customers to delay decisions. This is especially enabled by short feedback loops and continuous integration. From organization perspective Agile approach creates many added values since it enforce creating self-organizing and cross-functional teams capable to develop the whole software, from systemization via coding to functional testing. Finally, Agile requires everyone to be involved creating the big cultural change.

REFERENCES