Chapter 1

Communication in Agile Software Development

Vadim Costache

1.1 Introduction

Communication is one of the hottest issues of our time. This chapter is concerned with the communication within the agile software development paradigm.

In this context, we are going to discuss a paradigm change: The original view on communication has been that it is a means to transmit information from a sender to a receiver. Modern theories on communication see it as an organic and dynamic concept and compound of our lives, as the modifier of meaning, of our reality and implicitly of the organizations we are part of and how we work.
1.2 Interpersonal Communication

1.2.1 Four Square Model

Firstly, let’s observe the structure of human utterance: Friedemann Schulz von Thun has theorised the four sides model of communication\[12\]. It describes four layers, as in Figure 1.1:

1. The matter layer (what one informs about - the brute information)
2. The self-revealing layer (what one reveals about oneself)
3. The relationship layer (what one thinks about the person one speaks with)
4. The appeal layer (what one wants the other to do)

This model is an important foundation on which we formulate messages and feedback. In a multicultural organization, awareness of the differences of views, values and concepts is essential to formulating the right message. The Four Sides Model is a concept within a whole trend of shifting the concept of interpersonal communication.
1.2 Interpersonal Communication

1.2.2 The perspective on communication, from classical to modern

The classical (Transmission) communication model has been of treating communication as a simple means of transmitting messages from a sender to a receiver, and the personalities, organizations and events as ‘found’ things[2, p. 10].

In the 1970’s, Barnett W. Pearce and Vernon E. Cronen have begun working on a practical theory on communication that received the name ‘Coordinated Management of Meaning’ (CMM). It is a language for describing social interaction - through concepts, terms and diagrams rather than through propositions and theorems[2, p. 14]. It uses a ‘social constructionist’[2, p. 10] model of the world: CMM focuses on communication, because it is the primary social reality[2, p. 10]. It suggests that our communication, through content, structure and context shapes how we feel about ourselves and the people around us, that it ‘creates, sustains and/or destroys relationships, organizations and communities’[2, p. 24].

We can conclude that not only what we communicate matters, but in a great measure, how we communicate is of vital importance. Form and context affect meaning and through managing the four layers of communication theorised by Von Thun, we manage the relationships with other people and the comprehensibility of the information passed on. Interpersonal communication is the essential factor on which we build an organization. Individuals are the smallest units from which we form it. Cohesion and structure are based on communication at both interpersonal and organizational levels and people have an ever-increasing role in shaping and directing the organization.
1.3 Organizational Communication

Just like interpersonal communication, organizational communication is a concept that is currently experiencing a paradigm shift. The view on this matter evolved from seeing it as the means to move messages around the organization, to the means to shaping its entire structure.\textsuperscript{[9]}

There are two major approaches to explaining organizational communication\textsuperscript{[13]}:

1. **The container approach**, which assumes that the organization and the organizational communication are two different things, and that the former is the container for the latter.

2. **The social constructionist approach**, which assumes that organizations are shaped and built from and upon communication. Different choices for communication patterns define different structures and flows of communication within an organization. In the aid of this statement, we will consider the following patterns:

   (a) **Chain** - top-down approach, specific to the military and some business organizations.

   (b) **Wheel** (a.k.a. Star) - It represents the typical autocratic organization: all communication flows through the leader.

   (c) **The Y** - a combination between the Wheel and the Chain: has a central point and one of the ‘arms’ is longer.

   (d) **Circle** - As the name says, communication is circular - one member can directly communicate with his ‘neighbors’ on the circle. To the others, he can communicate indirectly, through his ‘neighbors’.

   (e) **All-Channel** - Analogous to free-flow communication, it can be considered as a circle with a full mesh of links between the members.
There are many ways to structure the organization. This raises the question of which communication structure is the best. To this end, Alex Bavelas and Harold Leavitt, two American psychologists, have devised an experiment\[4\] to measure the effects of the structure of communication on task performance:\[4\]:

The setup: The tested structures are the wheel, the Y, the chain and the circle. Five people participate. Each one is put in a coloured cubicle that has a number of slots, according to the communication pattern. No one knows the layout, nor their position within it. They are allowed to exchange as many messages as they want, on coloured sheets of paper. The same set of subjects plays the game in the same positions 15 times.

The objective: Each person is given a set of 5 symbols from a set of 6. The objective is for each player to find out which symbol they all have in common. The cubicles they are in have 6 switches matching the symbols. Each participant must press the switch for the symbol he thinks is the common one. If one incorrect switch is turned before the end of the game, it is considered an error.

The results:

1. Time: The wheel and Y were much faster than the chain and the circle
2. Messages: The wheel and Y used the least number of messages. The circle used the most.
3. Errors: The wheel and chain made the fewest errors, while the circle made the most. Still, the circle had the most error corrections.
4. Leadership: Statistical opinion on whether the group had a leader was, in decreasing order: wheel, Y, chain, circle. Agreement on who was the leader decreased in the same order (100% for the wheel).
5. **Improvement**: The people in the circle were likely to say that they could have improved their performance, while the ones in the wheel layout said that they could not improve much.

The initial experiment led to the conclusion that the more centralized the system, the greater the efficiency - the organization develops rapidly around the central positions.

However, Bavelas speculated that there is one large aspect at play, that was not covered by this experiment: the people and their morale. Lower morale could lower the accuracy and speed of the organizations employing the centralized patterns.

"[Bavelas]...suggested that the ability to evaluate partial results, to look at alternatives, and to restructure problems fell off rapidly when one person was able to assume a more central (that is, more controlling) position in the information flow. For example, insight into a problem requiring change would be less effective in the Wheel and the Star than in the Circle or the Chain because of the “bottlenecking” effect of data control by central members.”[4]

Another experiment was devised by Harold Leavitt in order to conclude which combination of direction and duplex is the fastest. The following conclusions were drawn[7]:

1. One-way communication is faster than two-way communication.
2. Two-way communication is more accurate than one-way communication.
3. Receivers are more sure of themselves and make more correct judgments of how right or wrong they are in the two-way system.
4. The sender feels psychologically under attack in the two-way system, because his receivers pick up his mistakes and oversights and point them out to him.

5. The two-way method is relatively noisier and looks more disorderly. The one-way method, on the other hand, appears neat and efficient to an outside observer.

From this experiment we can devise the conclusion that one-way communication is best in short-term critical situations when there is no time for debate, while two way communication not only offers better accuracy, but also maintains (or even increases) the morale of the team.

We can conclude this by stating that decentralized networks are better suited for situations of constant change, as lateral communication is vital: The members can thus communicate the changes they perceive, contribute with knowledge and ideas and give constant feedback.

1.4 Communication in an Agile environment

1.4.1 Who needs communication?

The stereotype on programmers is that they lack the ability to communicate. This is about to change in various areas of software development, where small, efficient teams will be needed.

We can keep in mind a parallel of commando troops versus the regular army: The regular army is based on a top-down ‘Chain’ hierarchy, while the commando troop communicates in an ‘All-Channel’ manner - intensive communication, all the team members have equal decisional rights (even if there is a team leader), high adaptability to unpredictable situations.
This section is mostly based on Alistair Cockburn’s book[5]. This is because this paper is mostly based on the books[5][3] on agile software development by Alistair Cockburn and Kent Beck. Although they touch similar aspects on agile methodologies, Cockburn is more people and communication-oriented(and better discusses the pros and cons of agile methodologies, analyzes the performance and economical aspects of growing the team), while Beck focuses more on the planning, administrative aspects(as well as the issues that might come up in the process of development).

1.4.2 Levels of listening

According to Alistair Cockburn’s book ‘Agile Software Development[5], shared experience between two people is a key ingredient to their successful and efficient communication. Conceptual constructs in their communication will then be built starting from a common point, rather than from scratch. As their common experience extends, they require an ever-simpler protocol to bring each other to date with what they want to communicate, saving time and effort.

Cockburn[5] uses the concept of Shu-Ha-Ri[6] from Aikido as a guide to classify the levels of mastery of skills and crafts (in particular, programming and communication). Whenever people pick up a skill, they traverse three levels of mastery: following, detaching and fluent:

1. Following - at this level, people learn and understand one single procedure that works (Shu means to keep, protect or maintain: the student learns and repeats a few techniques until he masters them)

2. Detaching - now, they discover 10 alternative solutions and analyze the advantages and disadvantages of each (Ha means to detach. ‘The student reflects on the meaning and purpose of everything that he has learned’[5].)
1.4 Communication in an Agile environment

3. Fluent - at this stage, people don’t make the difference between procedures, as they intertwine them with each other and with new ones without thinking. (Ri means to transcend. The student now becomes a practitioner)

This whole concept is a proposal of communication with oneself and discipline. It will also serve us ahead, when talking about the communication between team mates in general and in pair programming.

1.4.3 A Cooperative Game

First, though, let’s discuss the types of games: zero-sum, positional, cooperative, finite and infinite\[^5\]. They can be classified as in Figure 1.2:

![Figure 1.2: Different categories of games - Agile Software Development\[^5\], page 30](of games.png)

The games at play in the context we’re discussing are finite goal-directed cooperative games (software development) and infinite task-oriented competitive games (career management and organizational survival).
The metaphor (he puts it above the status of metaphor and calls it a comparison partner) Alistair Cockburn has chosen for describing the ‘game’ of software development is rock climbing. He sees it as a game of close team cooperation and shared responsibility that balances safety, challenges and risk.

The other games are career management and organizational survival. And they are contradictory, and both are pulling the ‘game’ of software development in different directions: The programmers might take actions that damage the development game, but help out their own. On the opposite side, the company is striving to grow. One project is only one move in the larger dimension of the company’s game. Sometimes, projects can be sabotaged by their own managers for strategic reasons. By doing this, they affect the careers of the programmers.

As opposed to the profit-oriented game, Cockburn presents Open Source Development[5, p. 40] as an infinite, non-goal-directed game: people work on these projects for fun, not for profit or career.

1.4.4 The people in an agile development team

In his book Agile Software Development[5], Alistair Cockburn states that a team that functions well is able to perform well, whether or not technologies involved change, help or hinder them[5, p. 44]

As long as the system is comprised of people, understanding how they work, what makes them perform better and what are their limits is essential to the well-being of the organization that they work for.

Humans are unpredictable and spontaneous. Their personal life and style, their beliefs and values differ from one another. This combination of personality and unpredictability gives way to great potential in a team
and must be used creatively. Communication in an agile context relies on the entire team and its communication with other teams. Good solutions may come up from the least expected people. Different tasks will be done better and with more enthusiasm by different people. This can also lead to a degree of specialization of individuals, while still keeping track of the entire project through the Agile practices. This eventually creates a benefic situation for the members of the team, the team as a whole and the organization.[5, p. 48]

People are error prone and avoid risks

Another aspect of people is that they make mistakes. This is predictable: mistakes will be done and there are ways (such as unit testing and debugging) to keep them at bay. The not-so-obvious thing is HOW they deal with their mistakes: They prefer to fail conservatively, rather than risk succeeding and have difficulty changing working habits[5, p. 49].

The choice to fail conservatively is explained through a quotation by Cockburn of Massimo Piattelli-Palmarini[5, p. 49]:

"people generally are risk-averse when they have something in their hands that they might lose and risk-accepting if they are in the process of losing something and may have a chance to regain it”[10, p. 58]

This affirmation is based on a dual experiment done by Palmarini:

1. People are given 300$ and then have to choose between a guaranteed 100$ more or a 50/50 chance at 200$ more. They prefer taking the guaranteed 100$.

2. People are given 500$ and then have to choose between having 100$ taken away from them or a 50/50
chance of having 200$ taken away from them. They will prefer to risk having 200$ taken from them.

This experiment explains why, for example, some managers choose not to use agile methodologies, even if it greatly outperforms a traditional approach that already worked. On the other side, there’s a chance that a new and revolutionary approach is employed by people who don’t have anything to lose if the project fails. This issue is also coupled with the tendency to maintain habits. Another effect is the tendency to reinvent the wheel: programmers will choose to write their own solutions to problems that have been previously solved. This is described as the Not-Invented-Here (NIH) syndrome. This can be countered with tolerance and discipline. Most of the time, discipline is the weak spot of methodologies that deal with these issues. According to Cockburn, from all methodologies, Extreme Programming is one that sticks with the people:

”People report programming in pairs to be enjoyable. They therefore program in pairs quite happily, after they adapt to each other’s quirks. While programming in pairs, they find it easier to talk each other into writing the test cases and adhere to the coding standards.”[5, p. 52]

Another reported benefit of pair programming is the fact that the steepness of the learning curve is lowered when programmers are set in pairs. Also, code is produced better and faster.

Rewards

Creating a reward system for a team can be tricky. Cockburn states that rewards actually reduce the joy and productivity of activities (even of the fun ones). In the aid of this affirmation, Cockburn quotes from Alfie Kohn’s book ‘Punished by Rewards’[5, p. 58]:
‘Young children who are rewarded for drawing are less likely to draw on their own than are children who draw just for the fun of it. Teenagers offered rewards for just playing word games enjoy the games less and do not do as well as those who play with no rewards. Employees who are praised for meeting the manager’s expectations suffer a drop in motivation. […] In one study, girls in the fifth and sixth grades tutored younger children much less effectively if they were promised free movie tickets for teaching well. The study, by James Gabarino, now president of Chicago’s Erikson Institute for Advanced Studies in Child Development, showed that tutors working for the reward took longer to communicate ideas, got frustrated more easily, and did a poorer job in the end than those who were not rewarded’.

Cockburn’s conclusion [5, p. 58] is that pride in work, pride in accomplishment and pride in contribution are the rewards that retain a person’s intrinsic motivation. This lead him to develop a strategy of small wins that keeps up the pride and morale of the team: “simplest first, worst second”.

By constructing a “walking skeleton”, a barely connected version of the system that can handle just one small type of action, the team learns how to work together and gains an early win.[5, p. 59]

Early wins will add cohesion and confidence to the team. As a result, they will now be able to tackle the hardest problems. If the team is not ready for the hardest problem, the hardest problem that can surely be solved by them is to be chosen. [5, p. 59]
1. **Communication in Agile Software Development**

1.4 Drawing on success within the agile team

There is a number of human characteristics that can be put to good use [5, p. 62]:

1. Being able to learn - novice users are more easily adaptable to agile methodologies. Also, they can be helped to learn within the project by splitting it into subprojects

2. Being malleable - people act differently, given different motives and new information.

3. Being good at looking around - they are very good at observing inaccuracies and errors in other’s code. This ability is exploited by the concept of collective ownership: everybody can change any part of code in the project.

4. Taking pride in work

5. Liking to be a good citizen (of the company, of the team)

6. Taking initiative

1.4.5 The team

In this section, we will follow Cockburn’s [5] view on team communication and how to set up the environment in order to get the best out of an agile team.

**Convection currents of information**

The way that the work environment is set up directly affects the communication of the team and implicitly how fast they solve problems. Keeping programmers in pairs and close to each other help them communicate without interrupting their focus on work and reduces cost for the project. For this, Cockburn [5, p. 72] uses the concept of **erg-seconds** (Ergs is a unit of work (or energy spent)) to capture the cost
1.4 Communication in an Agile environment

of energy spent and the time lost to get an answer to a question and the metaphor of gas dispersion to explain how, by keeping closeness inside a team and between interdependent teams, the time and energy wasted to solve a problem is minimized. This is described in Figure 1.3:

Another concept, that of osmotic communication [5, p. 73] is introduced: When people perform their work, they unconsciously pick up traces of the communication around them. If there is anything relevant that goes past them, they can immediately pick it up and listen to solutions or join discussions. So, the closer the people in a team are, the better they communicate and the more information they re-
receives. There is a downside to this concept: it works the other way around when unrelated teams are placed closely together - the communication of one team disrupts the communication of the other one.

Information radiators

Communication can be performed indirectly by placing so-called information radiators\[5, p. 76\] at places where they are visible and useful. They display information in places where passers by can see them. They can be cleverly placed to provide important information and avoid the consumption of erg-seconds in certain situations. For example, they can be used as progress reports. Cockburn states\[5, p. 76\] that hallways qualify very nicely as good places for information radiators, but web pages don’t, as they require more effort than most people are willing to expend.

Caves and Commons

Starting from the theory of hot air\[5, p. 79\], that shows the usefulness of information flow hotspots, such as the coffee machines or hallways, one step would be to add whiteboards in these places. The next one is to improve the workspace to contain as much information and help communication as much as possible. Cockburn\[5, p. 80\] proposes the caves and commons layout - depicted in Figure 1.4.

Jumping communication gaps

For the best communication between programmers, they should be able to communicate as ample and complex as they need. The ideal is to have them close to each other and allow them to use all the means of communication:

1. Physical proximity
2. Three-dimensionality - direct sight, instead of two-dimensional video communication

3. Smell - it is a sense that helps pick up more context from one person to another.

4. Kinaesthetics (sensation of movement)

5. Touch

6. Visuals - the gestures

7. Cross-modality timing - the timed correlation of all the above senses, between two people communicating to each other.

8. Low latency - during direct communication, information can be processed more easily and answers come back quicker.
All these lead to efficient communication, builds trust and helps learning inside a team. Cockburn[5, p. 84] categorizes the completeness and effectiveness of communication by a gradient from cold to hot, as depicted in Figure 1.5.

![Figure 1.5: Communication effectiveness[5, p. 84]]

**Teams as communities**

We have discussed how a team can communicate well, but not much on how to motivate them to communicate at all. The personal goals or interpretations of the common goals affect the speed with which the team reaches the goal(Figure 1.6).

Useful methods for keeping the motivation alive are the **project priority chart**, which Cockburn[5, p. 88] considers simple and vital mechanism to use for aligning the effort of all the members of the team, a **balance of amicability and conflict** and **citizenship**. Small-scale conflict is good for a team: it can generate constructive debate and exchange of opinions. Citizenship is a matter of being fair and respectful towards the others. This includes[5, p. 90]:

1. Getting to meetings on time
2. Answering questions from other people
3. Bothering to mention things that one notices
1.4 Communication in an Agile environment

Figure 1.6: Two depictions of members of the team pulling the project in different directions. (b.) shows a team that is better aligned towards the goal.[5] - p. 88]

4. Following group coding conventions

5. Using code libraries

“Citizenship permits programmers who disagree on coding styles to nonetheless create a common coding standard to themselves.”[5] p. 90

Strengthening the team with wins

One of the best ways of building cohesion, trust and morale in a team is by having them be successful in producing results, no matter how small.

1.4.6 Feedback

The most important aspect of the agile paradigm is feedback. We can distinguish three feedback loops within the process of agile development of software:
1. Pair Programming

2. Unit Tests

3. Continuous integration

**Pair Programming**

By programming in pairs and switching pairs, code is automatically reviewed by another set of eyes with a fresh, unbiased perspective. Pair programming also helps the members of the team to motivate each other to write thorough unit tests.

Jeff Anderson, an agile team leader, has conducted an experiment on pair programming: the ‘observer’ had to ring a bell every time he helped the ‘doer’. The average time between bell rings on each pair has been within the 10-30 second range - thus revealing the usefulness of pair programming[1]

**Test Driven Development**

Following the recommended agile practices for TDD(Test Drive Development) makes sure that you receive feedback from the specifications you’ve written yourself: Test, code and then design[1]

**Continuous Integration**

Integration of the code in the entire code base may reveal hidden errors that unit testing did not reveal. It synchronizes the entire team and helps it work together on the incompatibilities between code sections.
1.5 Conclusion

Client Feedback

The three feedback loops happen within the development team. There is one extremely important aspect left to be discussed: the feedback of the client. No matter how well the agile team is developing, without constant input from the client, the direction of the software in development may easily drift away from the client’s needs. Matt Stine, an group leader and article author for dzone.com, uses the metaphor of driving a car: constant steering is needed to keep it on the road. Frequent demos of the software not give the client insight on the progress and the direction of the software[11].

1.5 Conclusion

In this chapter, we’ve gotten a rough idea on interpersonal communication and how it affects the shape of the organization. Ultimately, we’ve come to see how this has lead to the concepts of agile development and the communication within an agile team.

Agile methodologies work well with small, commando-style teams that communicate intensely and adapt to the requirements along the way. Beyond being a human-friendly approach that gives satisfaction to the team members, it is also economically beneficial to the organization employing it. Such a small team can sometimes do a better job than a much larger one, that needs a lot of overhead for communication and administration.

Even though it is acknowledged that agile methodologies do not work for very large teams[5], we must keep in mind that two way communication and transparency must be kept within the layers of an organization in order to preserve the most important factor: the human one. Through better employee satisfaction and involvement, the organization obtains better productivity and smaller costs.
Bibliography


