Organisational Culture in Agile Software Development

Peter Wendorff

ASSET GmbH, Am Flasdieck 5,
46147 Oberhausen, Germany
P.Wendorff@t-online.de

Abstract. Recently a number of so-called "agile" software development methods have been proposed. Interestingly, these approaches have been met with "both enthusiastic support and equally vigorous criticism" among experts in the field. At present the software engineering community is split, and seemingly irreconcilable "schools of thought" have emerged. In this paper we identify an important characteristic of any software engineering method: its set of tacit basic assumptions. We retrieve some important basic assumption that underly agile software development and discuss an example to illustrate in detail how conflicting basic assumptions can lead to fundamental disagreement about software development methods.

1 Introduction

A software development method (SDM) provides a prescriptive, systematic, and explicit description of resources, activities, and artefacts in order to produce software. In the late 1990s a number of so-called "agile" SDMs have been proposed, for example "Extreme Programming" (XP) [2], the "Crystal" family [5], or "Adaptive Software Development" (ASD) [7]. They claim to be superior to other methods in some situations that are characterised by vague requirements and rapid change. They share a core of values and principles published as the "Manifesto for Agile Software Development" on the World Wide Web [1]. XP is by far the most widely used agile SDM at the moment, and even the first ever dynabook of the Institute of Electrical and Electronics Engineers (IEEE) has been devoted to it [9].

Agile SDMs have quickly gained a remarkable degree of acceptance in parts of the software engineering community. Interestingly, they have provoked a vivid and often controversial exchange of opinions, for example, published in the dynabook mentioned above. The observation by Jawed Siddiqi of "both enthusiastic support and equally vigorous criticism" of XP in the dynabook [9] extends to agile SDMs in general [8].

Highsmith introduces the term "rigorous" software development method [8] for most methods that do not explicitly focus on agility, and we will adopt his terminology in this paper.

There is yet no convincing empirical evidence that agile SDMs outperform other approaches, but there is equally little empirical evidence to suggest the opposite. This

situation of uncertainty calls for an unprejudiced discussion of agile methods, but the debate has become weirdly polarised and opinionated in many instances (cf. [9]).

Jim Highsmith makes an important point when he remarks: "Many of the debates about Agile versus rigorous practices have no basis in fact - they are purely emotional and based on one's culture, one's values and beliefs. Now, emotion-based reactions are at least as valid as fact-based ones, but they do tend to create high-volume rhetoric" [8, p. 167].

We agree with Highsmith's view that much of the debate aboute agile software development is due to cultural differences. We believe that in order to understand and appreciate any software development method it is necessary to understand its underlying culture. The cultural perspective illuminates the influence of conflicting, unconscious basic assumptions as a primary source of disagreement over different software development methods. Basic assumptions derive much of their power from the fact that they operate outside awareness. They are the sublime result of complex and prolonged learning processes and therefore difficult to detect and decipher.

Agile software development is based on some basic assumptions that seem to contradict the basic assumptions on which many other approaches are based. We believe that a clear elaboration of these basic assumptions is a necessary prerequiste for an unprejudiced discussion of agile software development within the whole software engineering community. The aim of this paper is to contribute to this process of elaboration by suggesting some basic assumptions of agile software development.

In section 2 we will present a brief description of organisational culture and the model developed by Schein, that will be used as conceptual framework in subsequent sections. In section 3 we will elaborate some basic assumptions that underly agile software development.

2 Organisational Culture

The concept of organisational culture has attracted much attention from scholars as well as managers recently. Much of this interest has been due to the apparent failure of traditional organisational analysis to explain phenomena in organisations based on objective and formal structural characteristics.

The culture perspective uses established ideas from fields like anthropology, psychology, and sociology. One of its central assumptions is, that organisations cannot be understood comprehensively in terms of their formal characteristics alone, but that there exist influential informal elements in organisations. Informal elements within organisations may include myths, heroes, traditions, mental models, animosi-ties, patterns of behaviour, social perceptions, bias, groupthink, group dynamics, politics, coalitions, friendship, revenge, etc. The culture perspective emphasises the importance of these informal aspects of organisations for the analysis of organisa-tional life. Much of the informality in organisations results from the fact that their members are human beings, and that human behaviour defies a definition in purely formal and rational terms. Therefore, the analysis of processes in organisations should take these informal aspects into account.

2.1 Elements of Organisational Culture

A look at contemporary textbooks on management shows, that there is no single, universally accepted definition of organisational culture. Nevertheless, most defini-tions refer to some points in the following definition:

Organisational culture refers to

- a common set of beliefs, attitudes, perceptions, assumptions, and values,
- that is shared by the majority of an organisation's members,
- where it is clearly observable who shares in the culture and who does not,
- while it reflects accumulated common learning by organisational members,
- who develop it as a response to perceived internal or external requirements,
- regarding it as a valid source of appropriate explanations for relevant situations,
- for which it suggests or prescribes a range of acceptable behaviour,
- that is taught to new members to facilitate their understanding and integration,
- and is therefore stable and persistent over long periods of time.

As organisational culture is a theoretical artefact, that can only be measured indirectly, it is important to identify elements of an organisation that indicate its culture. A popular layered conceptualisation of organisational culture has been introduced by Schein [10], who differentiates three levels of manifestation. At the first level there are "artefacts", that are most easily discernible by an observer. At the second level there are "espoused values", that are more difficult to observe. The third level are "basic assumptions", that are most difficult to detect and express. We will now briefly explain these three levels.

Artefacts are the most visible manifestations of organisational culture. Usually they are created by humans in order to solve a problem. Artefacts comprise rules, jargon, stories, symbols, office layouts, and ceremonies. These elements of a culture are visible to an outside observer directly by watching the behaviour of the organisational members.

Espoused Values are consciously held reasons for behaviour. They are strongly related to ethical codes, and they express what ought to be done. Values are not directly discernible for an outside observer, instead they can only be investigated indirectly through interpersonal communication.

Basic assumptions are the least obvious manifestation of culture. They comprise reasons for behaviour that are not consciously held by organisational members, because they are taken for granted. Basic assumptions are not confrontable or debatable, and examples include the basis on which individuals are respected, whether cooperation or competition is desirable as mode of behaviour, and how decisions are made. These unconsciously held assumptions guide human behaviour, and they are not directly observable for an outside person.

The relationship between the three levels of organisational culture described above can be demonstrated by the following example. We assume two basic assumptions, namely (a) humans are generally lazy and try to avoid effort, or (b) humans are generally motivated and enjoy to do good work. Depending on our basic assumption we could then proceed to the level of values and find that (a) strict control of procedures

is desirable, because it leads to productivity, or (b) a motivating workplace that provides opportunity is desirable, because it leads to productivity. Depending on these values, we might then choose an appropriate artefact to serve our value, for example, by (a) using a production line, or (b) forming a semi-autonomous work team.

2.2 Functions of Organisational Culture

The importance of culture for the smooth functioning of organisations has been emphasised by many writers. Organisations provide venues where people with differing backgrounds meet, and clearly this creates a potential for disagreement. In order to act effectively, an organisation has to achieve some degree of consensus and cooperation, for example, through formal regulations. This formal approach often results in organisational designs that rely on extrinsic motivation of organisational members. There is strong evidence that common and intrinsic motivation of members does often increase the effectiveness of organisations. An organisational culture represents a shared basic mindset, and therefore it can motivate action, facilitate agreement, and encourage cooperation. In this way it can lead to intrinsic motivation, thereby reducing the need for extrinsic motivation [4, pp. 89].

3 Organisational Culture in Agile Software Development

In February 2001 many of the leading inventors and proponents of agile SDMs met to identify common core elements of agile SDMs. This has led to the formulation and publication of the "Manifesto for Agile Software Development" on the World Wide Web [1]. An annotated version of the Manifesto can be found in [5]. This manifesto is still the most up-to-date and most comprehensive attempt to compile and publish the defining commonalities of different agile SDMs. The manifesto defines 4 values and 12 principles.

On the backcloth of Schein's layered conceptualisation of organisational culture the 12 principles from the manifesto clearly qualify as artefacts in Schein's hierarchy, while the 4 values from the manifesto are in fact espoused values according to Schein. But there is one layer in Schein's model of organisational culture that is not matched by any material provided in the Manifesto: the underlying basic assumptions of agile software development.

In the following subsections we will attempt to recover some basic assumptions underlying agile software engineering. We have mainly relied on the books by Highsmith [7], Cockburn [5], and Beck [2], as well as material from the manifesto.

The following six subsections all correspond to the same structural pattern. First, the headings of the subsections correspond to the six categories defined by Schein to categorise basic assumptions [10, pp. 94]. After a short introduction of the general category we narrow the context to a subject with particular relevance to our discussion. After that we present some references that illustrate some aspects of different agile SDMs regarding the given context. Then we present a brief summary of our deliberations, and finally we formulate a basic assumption.

Generally, our work does owe much credit to the book [8] by Jim Highsmith, in which he presents a very readable overview of many current agile SDMs as well as material on the biographies of their founders.

3.1 Assumptions About the Nature of Reality and Truth

Assumptions about reality are a cornerstone of any culture. The members of the culture share an understanding of what is real, how things are perceived, what is important and what is not, how to gather and use information, when and how to act, etc.

Context. An important distinction can be made due to different levels of reality. For example, external physical reality can be determined empirically by objective tests, while subjective perceptions shared by a group of people, that cannot be tested empirically, are referred to as social reality. Not all questions concerning reality and truth can be answered at the level of external physical reality, and indeed it is one of the important functions of culture to provide orientation in these cases where objective tests are impossible or too difficult to construct [10, pp. 97].

References. Fenton and Pfleeger use a quotation from a popular book by DeMarco, "You cannot control what you cannot measure" [6, p. 11] to express their belief that classical engineering techniques like scientific measurement should be adopted in software engineering. They explain: "Even when a project is not in trouble, measurement is not only useful but necessary. After all, how can you tell if your project is healthy if you have no measures of its health? So measurement is needed at least for assessing the status of your projects, products, processes, and resources" [6, p. 11]. In their book, "Software Metrics - A Rigorous and Practical Approach", they describe the quest in rigorous software development for objective definitions of software quality attributes like size, structure, complexity, understandability, etc. Unfortunately, there is good reason to believe that these objective definitions are not feasible, because concepts like complexity are in fact highly subjective, and for example, often one person's simplicity is another person's complexity [6]. Many software developers are quite opionionated regarding software quality attributes, and this is a constant source of disagreement in software projects.

Highsmith notes that in agile software development decision making is generally based on power sharing. If decision-making authority is delegated, then it is granted from below, not from above [7, pp. 214]. Different opinions must be reconciled through compromise, and Highsmith regards the willingness to compromise as necessary behaviour of developers. He notes that, for example, in the field of software quality attributes trade-offs are often unavoidable, and that these necessitate the involvement of relevant stakeholders. Given the fuzzy nature of software quality attributes he concedes that, "compromising on values and beliefs is much stickier" [7, p. 217]. Highsmith proposes three types of compromise, namely synergy, mutual concession, and appeasement. Obviously, all of these three types of compromise are primarily based on intensive social interaction among equals, not on coercive authority or authoritative science.

Cockburn observes, "on an effective team, the people pull approximately in the same direction. They actually all pull into slightly different directions, according to their personal goal, personal knowledge, stubbornness, and so on. They work together at times and against each other at times" [5, p. 99]. In order to increase alignment within the team he recommends "microtouch" intervention by the team leader, where a small increase in alignment of all team members can effect large changes in team performance. On the role of conflict within an organisation Cockburn notes that conflict is even desirable in some instances, for example in order to alert the team to design problems. Accordingly, he contemplates "the intentional use of small doses of conflict to get people to meet and learn to talk with each other" [5, p. 101].

Beck gives an important role to the concept of simplicity in XP, as it is one of the four values [2, pp. 30], one of the basic principles [2, p. 38], and one of the core practices [2, p. 54]. Beck does not provide any sophisticated, objective definition of the concept of simplicity, and the explanations he gives (cf. [2, p. 57 and p. 109]) are vague and questionable. So, how can simplicity be a meaningful concept in XP if there is no objective definition? Beck provides the answer referring to the levelling effect of communication: "The more you communicate, the clearer you can see exactly what needs to be done and the more confidence you have about what really doesn't need to be done" [2, p. 31]. The core practice of pair programming, where two developers sit in front of a single computer and work together, increases the mental alignment of the two programmers and thereby increases the mutual understanding of the two persons [2, pp. 66 and pp. 100].

The following practices taken from the Manifesto [1] relate to the assumptions about the nature of reality and truth in agile software development.

- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity -- the art of maximizing the amount of work not done -- is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.

These principles are based on terms like "working software", "technical excellence", "good design", "simplicity", and "best architectures, requirements, and designs". It is obvious, that there doesn't exist any consensus in the software engineering community about the definition of these terms. Nevertheless, because many crucial artefacts in the Manifesto rely on these notions, it is necessary that some consensus about these concepts exists, at least at the level of social reality.

Discussion. Rigorous software development has a low tolerance toward ambiguity, and the obvious solution is to adopt traditional engineering practices, e.g. scientific software measurement, to reduce ambiguity. Thus, consensus is established on the basis of scientific authority at the level of external physical reality. Nevertheless, this approach suffers from a number of inevitable drawbacks, and admittedly, many software measures that have been proposed are of questionable practical value [6]. The culture of agile software development is more tolerant toward ambiguity, and, for example, a concept like simplicity is obviously regarded as useful. In the absence of a convincing objective, scientific definition for such a concept agile methods rely on the emergence of consensus at the level of social reality. This desired consensus emerges over time as the result of intensive social interaction, and that is one of the reasons why agile methods try to sustain a high level of social interaction.

Basic Assumption. "Social interaction leads to consensus."

3.2 Assumptions About the Nature of Time

A "natural" notion of time is taken for granted in most cultures, yet there exist different assumptions about the nature of time in different cultures. One aspect of this is the perception of "being on time", obviously an important issue in software engineering that is traditionally focused on deadlines.

Context. A shared notion of time is important to synchronise activities in an organisation in an orderly way. If different assumptions about time exist within an organisation, tremendous problems can emerge. For example, in a biotechnology company serious communication problems developed because managers used a notion of time labelled "planning time", while scientists used a different notion of time labelled "development time" [10, p. 109].

References. Boehm, in his landmark book "Software Engineering Economics" [3], used classical engineering practices and ecomomic models as basis for a rigorous approach to software effort estimation. His work has influenced generations of managers and developers since then. It is fundamentally based on the notion of "monochronic time" [10, p. 107], where time is measured by hours, where a man-hour is a standardised measure of production capacity, and hitting a deadline is the ultimate measure of success. Effort is meticulously calculated in hours, and completion dates are set as precise calendar dates. Time is seen as a linear resource that is compartmentalised into appropriate assignments that are then completed according to plan. The COCOMO model developed by Boehm has later been refined, and it is one of the most popular frameworks for effort estimation and project planning in software development. COCOMO, as well as many other frameworks of its kind, heavily relies on data of past projects that are regarded as a valid predictor for future projects.

Highsmith contrasts workflow-oriented models that focus on tasks, as the basis of rigorous software engineering, to workstate-oriented models that focus on resulting

artefacts, as the basis of agile software development [7, pp. 235]. He continues: "The workstate approach says, 'Don't bother me with the detailed activities, just let me know when the work product (component) has reached a certain completion state' [7, p. 238]. He claims that a workstate-oriented approach works better in many areas of software development where "activities are concurrent, with partial completion and later refinement being the norm" [7, p. 239]. Concurrent development is an inherent management challenge in adaptive software development, that requires intensive interaction to synchronise activities, according to Highsmith.

Cockburn declares: "Software development is therefore a cooperative game of *invention and communication*. There is nothing in the game but people's ideas and the communication of those ideas to their colleagues and to the computer" [5, p. 28]. In a game the moves of players are not predetermined by a schedule, instead they are coordinated by interaction. Therefore, "the purpose of each activity is to move the game forward. Work products of every sort are sufficiently good as soon as they permit the next move" [5, p. 33]. Cockburn applies this viewpoint to the inevitable bottleneck activities of a software project and concludes that these are the areas where synchronisation of activities based on interaction promises huge performance gains: "The shifting bottlenecks in the system determine the use of overlapped work and 'sticky' information holders" [5, p. 201]. He regards this technique as valuable for any software engineering methodology and makes it a cornerstone of his own Crystal family of methodologies.

Beck denotes the planning technique used in XP "planning game" [2, p. 86]. The planning game is a highly interactive, incremental, and iterative technique used to match software requirements to development capacity. At any time plans can be revised in subsequent iterations as soon as deviations arise.

Discussion. One important function of time is the synchronisation of activities. The notion of time used in rigorous software engineering can appropriately be described as "monochronic time", for example implicitly assumed by Boehm. An alternative to monochronic time is "polychronic time" [10, p. 107]. If time is perceived as polychronic, then it is not seen as a linear resource that is divided into time units that can be matched to particular activities, instead several activities may run concurrently in order to accomplish a task at hand. The above references indicate that in agile software development monochronic time is seen as a less useful concept, instead the concept of time used is rather close to polychronic time, where synchronisation does preferably occur through interaction rather than a clock.

Basic Assumption. "Social interaction synchronises activities."

3.3 Assumptions About the Nature of Space

The use of space is highly visible in organisations and often it does have a powerful symbolic meaning. One obvious reason is that space is a scarce resource in most organisations, and therefore its allocation can symbolise the status of a person.

Context. An open-plan office may stimulate communication, but it limits the degree of privacy, and the opportunity for individual expression. A private office, on the other hand, provides a high degree of privacy and enables more individual expression, but it may become a barrier for spontaneous communication [10, pp. 115].

References. Highsmith stresses the role of a shared work space for a team: "A factor contributing to adaptive project success is shared work space - a war room or team meeting place. [...] Adaptive teams need a team-owned place in cyberspace where the team can share context and content, where team members can interact one-on-one or in groups, where information can be both public and private, where there is an element of both work and play - a comfortable site to visit and to use." [7, p. 277]

Cockburn remarks that larger teams are often split into groups, and the groups are then assigned to different, scattered offices. This is a frequent source of problems because "each group forms its own community and usually complains about the other group. The chitchat in the osmotic communication is filled with these complaints, interfering with the ability of people in each group to work with each other in an amicable way" [5, p. 82]

Beck recommends an open workspace that combines a common workspace with small private spaces for XP. The reason for this arrangement is that, "XP is a communal software development discipline" [2, p. 79].

Discussion. The above references indicate that in agile software development the work space is not merely a place where people perform their professional duties. Instead it is rather seen as a venue where many different social functions, professional as well as private, take place. This is also reflected in the idea expressed by Cockburn and Beck that the workspace should provide room where people can socialise or prepare food. This deliberate integration of social needs into the professional environment is not accidental, instead it is intended to encourage communal life that leads to a cohesive team and effective communication.

Basic Assumption. "The workspace is a social venue."

3.4 Assumptions About the Nature of Humans

Every culture conveys assumptions about the nature of humans, for example about mission, motivation, ability, etc. The view that is hold about human nature inevitably has a strong influence on the fabric of society and organisations.

Context. The efforts of an organisation to effect certain behaviour will naturally be based on its prevailing set of assumptions about human nature, for example, its control and reward systems will be designed accordingly [10, p. 123].

References. Highsmith builds his theory of adaptive software development around the themes of self-organisation, emergence, and collaboration. In his discussion of effective collaboration he refers to attitudes like trust, respect, participation, and commitment [7, pp. 129]. He regards these attitudes as inherent in all humans to different degree and that the management challenge for an organisation is to activate this potential. He points out that the willingness of individuals to volunteer high performance in the workplace depends on the satisfaction of their own physical needs, emotional needs, and self-interest. Success requires both, ability and motivation [7, p. 133].

Cockburn describes common "failure modes" [5, pp. 48] and "success modes" [5, pp. 67], but he warns that these generalising statements do only apply to some degree to any individual [5, pp. 46]. It is the primary management task of an organisation create an environment where the success modes of individuals can take effect, and where the need to control their failure modes can be reduced [5, p. 73].

Beck devises in his book on XP to "Work with people's instincts, not against them" [2, p. 41]. Indeed, it is one of Becks claims that XP is a methodology that reconciles the instincts of programmers and the interests of the organisation [2, p. xviii].

Discussion. The above references indicate that in agile software development the individual is seen as a valuable human resource with high potential for productive work. The exploitation of this potential should not be taken for granted by an organisation, however, because it is ultimately at the discretion of the individual to put these capabilities to a productive use for the organisation. Therefore, an organisation must take human needs into consideration that go far beyond payment.

Basic Assumption. "Happy people do good work."

3.5 Assumptions About the Nature of Human Activity

Humans do not exist in isolation, instead they interact with their environment. A culture entails assumptions about the appropriate way of interaction with the environment for individuals and groups.

Context. At the organisational level a key question is whether members are encouraged to behave proactively, or whether they are assigned a rather passive role [10, pp. 127].

References. Highsmith states, "Speed is often the least risky course of action" [7, p. 203].

Cockburn, too, expresses a strong preference for action, and regards the willingness to take initiative as one of the success modes of humans. Referring to the other success modes of humans he continues, "With these, we see people taking initiative to get the job done every day, an ongoing activity that keeps the project operating at peak form" [5, p. 70].

Beck uses the metaphor of learning to drive a car to explain the management philosophy used by XP. The idea is to start early, act small, and prepare for corrections: "We need to control the development of software by making many small adjustments, not by making a few large adjustments, kind of like driving a car. This means that we will need the feedback to know when we are a little off, we will need many opportunities to make corrections, and we will have to be able to make those corrections at a reasonable cost" [2, p. 27].

Discussion. The above references indicate that agile software development is strongly biased in favour of action. In a situation where only incomplete information is available action is preferred to waiting for more complete information. The general attitude is that action is preferred to inaction. This may increase the possibility of undesired effects, but it is assumed that in these effects can be rolled back without difficulties. Early action usually results in higher risk, and therefore the importance of rapid feedback increases, to stop inappropriate action quickly.

Basic Assumption. "Action makes a project work."

3.6 Assumptions About the Nature of Human Relationships

A culture contains many assumptions about acceptable forms of human relationships. Important issues that must be addressed are the distribution of power and authority, and the nature of peer relationships.

Context. An important characteristic of culture is the dimension of "power distance" [10, pp. 132]. A fundamental question in this respect is the degree of power distance that is regarded as acceptable. If, for example, a low power distance is taken for granted in a culture then a manager may choose to substitute group-decisions for his own authority.

References. Highsmith notes that agile software development needs strong leadership: "While the following fact may seem paradoxical, adaptive environments require much stronger leaders than do deterministic ones" [7, p. 209]. Concerning the expectations of followers he adds: "Teams want a clear sense of direction and decisiveness from their leaders; they do not want arbitrariness or authoritarianism" [7, p. 210]. According to Highsmith the leader is empowered by the team to make decisions, and likewise the leader empowers the team members [7, p. 215].

Cockburn does give numerous examples of successful leadership in [5], but he does not provide a comprehensive discussion of leadership issues.

Beck vaguely describes the decision-making process in XP as "more like decentralized decision making than centralized control" [2, p. 72]. He has included a special role, called "coach", in XP: "What most folks think of as management is divided into two roles in XP: the coach and the tracker (these may or may not be filled by the same

person). [...] The measure of a coach is how few technical decisions he or she makes: The job is to get everybody else making good decisions" [2, p. 73]

Discussion. The above references indicate that agile software development is leadership-centric. A manager has formal authority granted by an organisation to act in certain situations, for example, to demand certain behaviour from others. A leader has the ability to influence the behaviour of others without using formal authority. The roles of leader and manager are independent, for example a manager may also act as leader in a certain situation. Generally, agile software development does not necessarily rely on strong formal authority, much in contrast to a traditional, hierarchical team structure with a single, appointed team manager who carries all responsibility. Nevertheless, in almost all published material about management in agile projects the figure of a strong and competent leader lurks underneath the egaliterian surface. This leader is usually described as very competent and successful. The relation between the leader and his followers is described as very harmonious and respectful. The impression is that in agile software development leaders are substituted for managers in a sucessful way, although it remains unclear why and how these leaders develop.

Basic Assumption. "Leaders influence followers."

4 Conclusion

Organisations provide venues where people meet, learn, and interact. Over time these social processes often result in organisational cultures that define "how things are done here". Schein has proposed a layered conceptualisation of organisational culture where basic assumptions are the least visible yet most influential element of the culture. Only if these tacit basic assumptions are uncovered it is possible to decipher and understand the observable behaviour of an organisation.

In this paper we have applied the perspective of organisational culture to agile software development. Using Schein's model we have retrieved the following six basic assumptions:

- Social interaction leads to consensus.
- Social interaction synchronises activities.
- The workspace is a social venue.
- Happy people do good work.
- Action makes a project work.
- Leaders influence followers.

We suppose that these basic assumptions are unconsciously held by many proponents of agile SDMs and are taken for granted by that community, rather than being debatable.

Basic assumptions refer to the most fundamental way we expect the world to be, and therefore they inform our thinking. We can think about them, we can discuss them, but we are extremely unlikely to change our own basic assumptions. They give us orientation in a complex world, and therefore, powerful, unconscious defence mechanisms fight anything that might violate or invalidate them. We treat a challenge of our basic assumptions as a threat to our identity. Because our defense mechanisms work unconsciously, we are often not even aware when they operate. This can make us vulnerable to stagnation, because we may reject new ideas not on the basis of a careful evaluation, but out of uncontrolled fear.

A look at the concept of simplicity in XP can be used to illustrate the importance of basic assumptions for our understanding. Assume, for example, a hypothetical software engineer who becomes interested in XP, buys a book on the subject, and starts reading. He will soon read about the core practices in XP, and simple design is one of these core practices, but he looks for an objective, operational definition of the concept in the book, he will probably be disappointed. He recognises that the core practice qualifies as a rule, which according to Schein is an artefact, but he cannot make much sense out of it, because a proper definition is missing. But why is this artefact there in the book, even in the form of a core practice, if it is indeed useless? Our hypothetical software engineer could proceed to the level of espoused values then, just to find that simplicity is indeed listed there. He may even agree with the value in the abstract, because simplicity sounds reasonable after all. Nevertheless, he possibly feels that he cannot accept the use of such a fuzzy concept without a reasonable definition. So he might get the impression that XP is disorganised and unscientific, because it relies on a concept that is not properly defined in an objective way. Without further consideration he might decide that he had enough of it and dismiss XP completely.

What is this engineer's basic assumption that unconsciously informed his thinking? In the best tradition of rigorous software engineering he is searching for an objective, scientific definition of the concept of simplicity. That is perfectly reasonable behaviour, even necessary, if software engineering is ever to become a true engineering discipline, he might say.

What is the corresponding basic assumption that informs XP as well as agile software development in general? It says, "Social interaction leads to consensus." In this statement "social interaction" is assumed to be trustful, collaborative, and competent interaction, of course. The term "leads to" refers to a process that takes some time before palpable results are obtained. The term "consensus" refers to social reality, basically it is assumed that the relevant team members align their notions of the concept of simplicity sufficiently, but it does not require that people outside the team agree. The result is a useful, barely sufficient means of communication in the context of the team.

It does become clear now, that the rejection of XP by our hypothetical engineer was caused by basic assumptions that he has held, that conflict with those that underly XP. For example, he is arguing at the level of external physical reality, whereas XP does only address the level of social reality in this case. This case shows that subtle differences in basic assumptions can only be detected when these basic assumptions are retrieved, elaborated, and subjected to analysis.

Agile software development is based on many basic assumptions that conflict with the school of rigorous software development. We think that this is the cause for the sometimes strong rejection of agile practices in the software engineering community. An open, unprejudiced discussion and appreciation of agile software development is, therefore, dependent on a thorough analysis of basic assumptions in both communities. In this paper we have presented some of the important basic assumptions in agile software development, but there exist many more.

References

1. The Agile Alliance: Manifesto for Agile Software Development. http://agilemanifesto.org/ (last visited on 16/06/2002), 2001.
2. Beck, K.: Extreme Programming Explained: Embrace Change. Longman Higher Education, 2000.
3. Boehm, B.W.: Software Engineering Economics. Prentice-Hall, 1981.
4. Brown, A.: Organisational Culture. Prentice-Hall, 1998.
5. Cockburn, A.: Agile Software Development. Pearson Education, 2001.
6. Fenton, N. and Pfleeger, S.L.: Software Metrics. International Thomson Computer Press, 1996.
7. Highsmith, J.A.: Adaptive Software Development. Dorset House Publishing, 2000.
8. Highsmith, J.A.: Agile Software Development Ecosystems. Pearson Education, 2002.
9. Institute of Electrical and Electronics Engineers: Dynabook on Extreme Programming. http://computer.org/seweb/dynabook/Index.htm (last visited on 16/06/2002), 2000.
10. Schein, E. H.: Organizational Culture and Leadership. Jossey-Bass Publishers, 1992.