Towards Better Understanding of Agile Values in Global Software Development[1]

Pär J Ågerfalk

Lero – The Irish Software Engineering Research Centre, Dept of Computer Science and Information Systems, University of Limerick, Ireland; and MELAB, Dept of Informatics (ESI), Örebro University, SE-701 82 Örebro, Sweden
par.agerfalk@ul.ie

Abstract. Globally distributed software development (GSD) and agile methods are two current and important trends in software and systems engineering. While agile methods seem to cope well with increasingly changing business environments, it is far from obvious how these light-weight processes can best contribute to GSD. In this paper, method rationale is proposed as an analytical tool to understand the values that underpin agile methods and how these map to the GSD domain. Specifically, the paper presents an initial analysis of the values and goals embraced by the 'agile manifesto' and compares briefly with partial results from an ongoing study on the use of agile methods in GSD.

1 Introduction

As part of the current trend towards globalization, offshoring and outsourcing, many software organizations are adopting global software development (GSD) as a new mode of software production [3]. GSD happens when software is developed by a geographically dispersed organization. For example, it is becoming increasingly common for US based companies to offshore routine activities, such as testing, to low-wage countries, such as India, as a way of cutting costs.

Another current trend in the software industry is the adoption of light-weight, agile software development processes. Agile methods, such as eXtreme Programming (XP) [4] and Scrum [12], operate on the principle of 'just enough method'. They emphasize the importance of people and prioritize working code over hefty documentation. Agile methods have evolved in response to increasingly changing business environments and volatile requirements and it may thus not be surprising that many organizations now look towards agile methods as a possible solution to some of the problems experienced in GSD.

However, there are many aspects of agile methods that seem to go directly against the GSD mode of software development. For example, agile methods typically emphasize face-to-face communication, on-site customers and aggressive release schedules. Distributed XP (DXP) [8] – an XP variant tailored for distributed

[1] This work has been financially supported by the Science Foundation Ireland Principal Investigator projects B4-STEP and Lero.

development – maintains, however, that certain XP principles are more susceptible to distance than others. We have also found in our own research that agile methods can indeed help reducing distance in GSD [7]. In order to create a solid foundation for further investigation, this paper explores an analytic framework based on method rationale. Method rationale has been proposed as a way of understanding an analysing the goals, values and choices upon which particular development methods are based [1]. In the context of this research we are particularly interested in understanding:

1. What values underpin agile methods and how are they operationalized into concrete method prescriptions?
2. How can these values be upheld in a GSD setting and how can they help reducing the negative impact of distance on GSD practise?

The paper proceeds as follows. Section 2 introduces the method rationale analytic framework. This is then used in Section 3 to characterise the rationale of agile methods through an analysis of the 'agile manifesto' and parts of XP as a representative example of an agile method. As a brief illustration, Section 4 then uses these characterizations as a basis for discussing one of the findings from an ongoing study [7]. The aim of this paper is to explore the usefulness of this approach to understanding the operationalization of agile values in method prescriptions and in actual development practice. The plan is then to broaden the study to include further agile methods and organizations, and more in-depth analyses.

2 An Analytical Framework based on Method Rationale

Our analytic framework is based on the concept of method rationale [1]. In this paper we extend previous work on the concept by introducing explicitly the actor as a framework component (see Fig. 1). As we shall se below, introducing the actor concept is key to facilitate analyses of rationality resonance [1, 13].

The framework in Fig. 1 consists of a number of components (depicted as UML classes): actor, value, goal, method fragment (in-concept and in-action), and a number of named associations between these. The framework draws on Max Weber's notion of practical rationality [14], which highlights that people, when acting socially, choose means in relation to ends, ends in relation to values and act in accordance with certain ethical principles. What this means for our framework is basically that:

1. Values dictate what goals are considered worthy of achieving.
2. Method fragments prescribe how to achieve goals.
3. Ethical principles dictate what method fragments are acceptable.

The term 'method fragment' refers to any part of a development method or a whole method, aligning with earlier method engineering research [5]. A method fragment is either 'in-concept' or 'in-action'. This is used to represent that a method fragment is either described as a prescription for action (in-concept), in, for example, a method handbook, or is a result of a method following action in practice (in-action) [1, 9].

As can be seen in Fig. 1, Value Rationale corresponds to (1) in the list above, and Goal Rationale corresponds to (2). With respect to (2), there is a distinction between a

goal that is the direct intention of a method fragment and higher level goals that the intention is a way of supporting. For example, the goal (intention) of the method fragment 'specify all user requirements in use cases' is probably 'all user requirements specified as use cases'. This, however, is only a means to achieve a higher level goal, such as 'all requirements specified unambiguously'. Hence, there is a Goal Achievement relationship between goals that represent how goals support other goals and thus form goal hierarchies or networks. The same principle applies also to values. This is in line with previous work on goal-oriented requirements engineering [10].

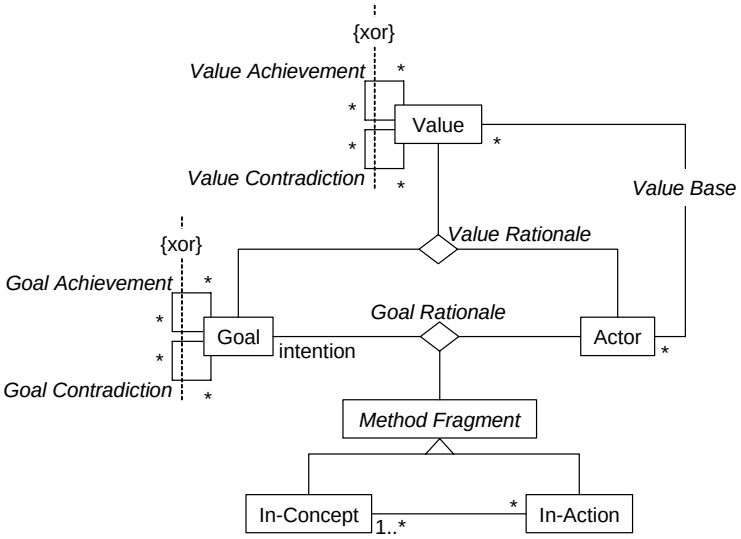

Fig. 1. Conceptual structure of method rationale.

The Value Base association represents the fact that not all values are operationalized as concrete goals but reside as guiding principles that the actor turns to as new situations emerge. Note that the Value Base is allowed to be empty since it represents only known explicit values. Depending on the stage of analysis, these may still have to be elicited. Rationality resonance occurs when two actors share the same method rationale, that is, when they agree to a common set of values, goals and method fragments.

3 Rationale of Agile Methods

The basic principles behind agile methods have been summarized by the leading agile method thinkers in what has become known as the 'agile manifesto'. The agile manifesto is presented as a set of values and associated principles, as shown in Table

1. Advocates of agile methods recognize the relevance of both sides of the value statements in Table 1 but choose to emphasize the first part of each statement more than the second. The values of the agile manifesto are all quite abstract, which is perhaps to expect of value statements that are to be used to judge the suitability of specific goals. The principles, on the other hand, are more specific and lend themselves to be treated as goals in the method rationale framework. The only principle that is indeed phrased more like a value statement than an achievable and measurable goal is that which refers to face-to-face conversations. Arguably, the first principle actually contains two goals with an achievement association.

Table 1. Agile values and principles (from www.agilemanifesto.org).

Values	• Individuals and interactions over processes and tools. • Working software over comprehensive documentation. • Customer collaboration over contract negotiation. • Responding to change over following a plan.
Principles	• Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. • Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage. • Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. • Business people and developers must work together daily throughout the project. • Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. • The most efficient and effective method of conveying information to and within a development team is face-to-face conversation. • Working software is the primary measure of progress. • Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. • 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. • At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

Altogether this gives us the following sets of agile values V = {v_1: Individuals and interactions over processes and tools, v_2: Working software over comprehensive documentation, v_3: Customer collaboration over contract negotiation, v_4: Responding to change over following a plan, v_5: Face-to-face conversations preferred} and goals G = {g_1: Customer Satisfied, g_2: Valuable software delivered early and continuously, g_3: Requirements changes are welcome, g_4: Working software delivered frequently, g_5: Business people and developers work together daily throughout the project, g_6: Motivated, supported and trusted individuals, g_7: Progress measured by working software, g_8: Sustainable development promoted, g_9: Continuous attention to technical excellence and good design, g_{10}: Simplicity is essential, g_{11}: Self-organizing teams, g_{12}: Self-reflecting teams}. When analysing the interrelationships between these

goals, we find that there are two clusters of goals all contributing to their own highest-level (or root) goal: one cluster aiming for customer satisfaction and another aiming for sustainable development – hence one externally oriented and one internally oriented. These clusters are depicted in Fig. 2 using plus signs to indicate goal contribution. We also find what is arguably a goal contradiction between g_3 and g_7 indicated by the minus sign.

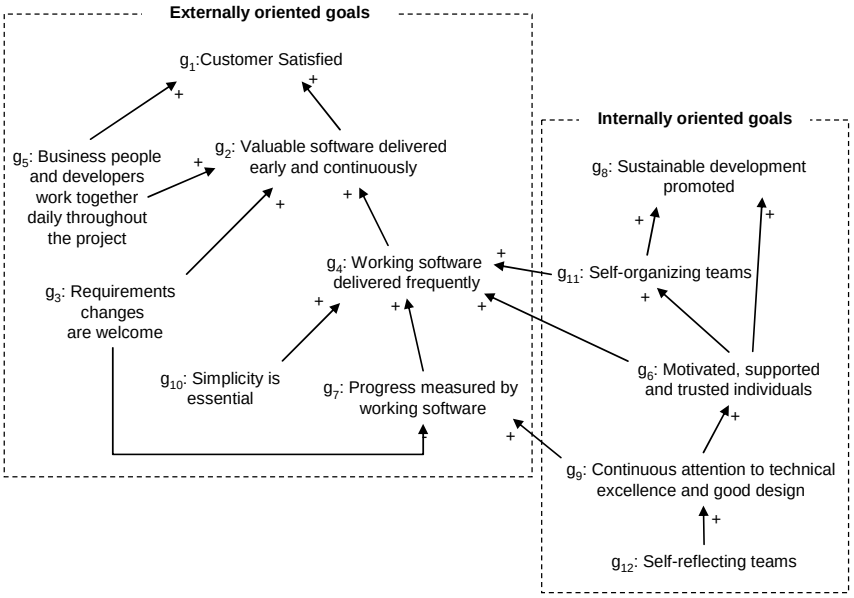

Fig. 2. Goal clusters in the agile manifesto.

The value rationale of the agile manifesto can be found by relating the identified goals G to the identified values V: *Value Rationale* = {(g_2, v_2), (g_3, v_4), (g_4, v_2), (g_5, v_3), (g_6, v_1), (g_7, v_2), (g_8, v_1), (g_9, v_1), (g_9, v_2), (g_{10}, v_4), (g_{11}, v_1), (g_2, v_1)}. Apparently, all values are operationalized into specific goals, except v_5, and all goals are grounded in at least one value, except g_1. The case of g_1 seems to indicate that there is indeed a non-expressed value of the agile manifesto: satisfied customers over …? In the case of v_5 it is unclear how the preference for face-to-face conversations is related to the other components of the agile manifesto. If it is treated as a value, there are no principles that operationalize it as a goal. If it is to be regarded as a goal, possibly grounded in v_1 it is singled out as the only goal that does not contribute to any other goal, besides g_1 and g_8. As such, it would thus be valued in its own right together with satisfied customers and sustainable development. The most obvious is perhaps to view it as a value with a value achievement association to v_1 – preferring face-to-face conversations can probably be traced back to valuing people and interaction over processes and tools. We also see that the two goal clusters identified above are mirrored by separate value clusters. This is not surprising and indicates that also the

values are oriented mainly internally or externally. It is important to remember that these goals and values are those expressed by the group of people behind the agile manifesto, who thus constitute the actor to whom this method rationale belongs.

In order to say something about goal rationale we clearly need to find specific agile method fragments that operationalize the goals of the agile manifesto. In order to do so, we pick XP as an example method. The reason for this is that XP is one of the most widely used agile methods, one that has been tried in a GSD context, and also one that was encountered in our empirical study [7]. As mentioned above, the work on 'Distributed eXtreme Programming' (DXP) [8] suggests that eight of the XP practices (small releases, metaphor, simple design, testing, refactoring, collective ownership, 40-hour week and coding standards) are independent of team locality and can thus be applied also in GSD. The remaining four practices (i.e., the planning game, pair programming, continuous integration, and on-site customer), on the other hand, depend on co-located team members and thus require tailoring. We will consider only these four XP practices in the remainder.

Interestingly, XP presents its own set of basic values, which are not expressed as prioritization pairs, but rather as ideals to strive for: communication, simplicity, feedback, and courage. Notably, the XP values are at a more detailed level than the agile manifesto ones. Nonetheless, they clearly are in the same spirit and map fairly well to the agile goals identified above. Fig. 3 shows how three of the four selected XP fragments map directly to goals of the agile manifesto.

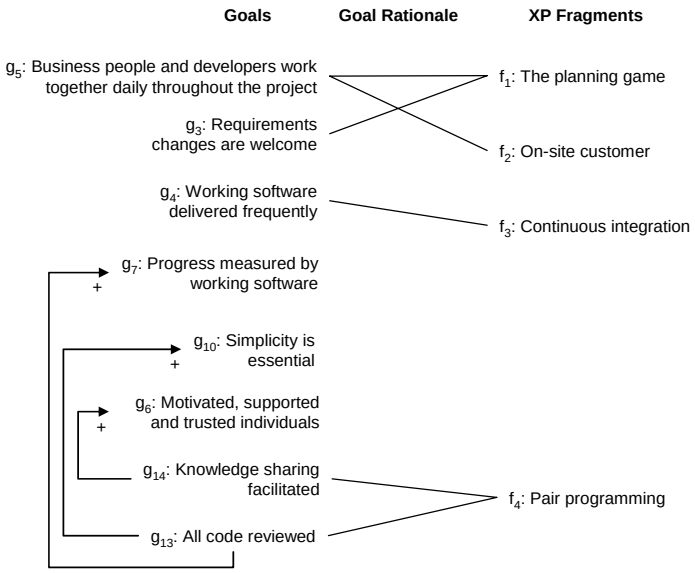

Fig. 3. Goal rationale in part of XP.

The fourth fragment, pair programming, does not map easily to any of these goals. However, the goals (or rather intentions) of pair programming (g_{13} and g_{14}) clearly

contribute to g_6, g_7 and g_{10} of the agile manifesto. Note that all four method fragments are in-concept, and are thus expressions of how the XP founders, most notably Kent Beck perhaps, envisage agile development. In the next section we will draw on an ongoing empirical study to illustrate how method rationale may play out in-action.

4 An Agile Method Fragment in GSD Action

The many challenges of GSD are, perhaps obviously related to the effects of increased distance between people. Distance is not only to do with spatial separation (a.k.a. geographical distance), but also with temporal distance (the dislocation in time experienced by two actors wishing to interact) and socio-cultural distance (actors' understanding of other actors' values and normative practices) [2]. Consequently, three high-level goals in GSD aim to reduce these distances:

g_{15}: Geographical distance minimized[2]
g_{16}: Temporal distance minimized
g_{17}: Socio-cultural distance minimized

Pair programming (f_4), the practice of always having two programmers work together on all production code, is a technique that intuitively would be difficult to practice in GSD. However, through time-shifting patterns, developers in our study [7] managed to create overlap and hence to reduce temporal distance (g_{16}). This way, an engineer in the US could work six hours a day paired up with another engineer in Belgium. It was believed that pair programming, in turn, helped increase knowledge sharing (as suggested by the in-concept rationale) which in turn was pointed out as an important contributor to reducing socio-cultural distance (g_{17}). Hence, not only did pair programming deliver the expected benefits. It also turned out to be the case that these benefits helped achieving GSD-related goals not part of the in-concept XP and agile manifesto rationale.

5 Discussion

This paper has presented an initial exploration of using the concept of method rationale for understanding better how agile values play out in GSD. The idea is to expand this study in several ways. First, the analysis of agile rationale needs to be performed in much more detail, covering all the sub-goals and different sub-method fragments to be found in a wider range of agile methods. Second, more in-depth case studies driven by questions generated by such an analysis are needed.

It is important to note the distinction between not adhering to a goal supported by a method and not knowing that the goal is supported [1]. The same is true for method

[2] Reducing geographical distance apparently contradicts the whole idea of GSD. However, there is currently significant interest in so-called nearshoring where low-cost but not so far offshoring locations are explored (such as US–Brazil and EU–Eastern Europe) [6].

fragments and values. It is thus important to understand what options were considered but dismissed, when considering someone's appreciation of a method in terms of method rationale. This aspect of method rationale has recently been put to the foreground in the work on evolutionary method engineering [11] as a way of documenting what options were considered when a method evolved in a project or organization. This kind of process-oriented method rationale [1] can help explaining why rationality resonance is not achieved, and the reasons for that – that is, why people disagree in particular situations. This is expected to be an important analytic device in the continuation of this research.

References

1. Ågerfalk, P. J., Fitzgerald, B.: Exploring the Concept of Method Rationale: A Conceptual Tool for Method Tailoring. In: K. Siau, (ed.) Advanced Topics in Database Research – Vol. 5. Idea Group (to appear)
2. Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., Ó Conchúir, E.: A Framework for Considering Opportunities and Threats in Distributed Software Development. In: Proceedings of the International Workshop on Distributed Software Development (DiSD 2005). Austrian Computer Society (2005) 47–61
3. Aspray, W., Mayadas, F., Vardi, M. Y.: Globalization and Offshoring of Software: A Report of the ACM Job Migration Task Force. Association for Computing Machinery (2006)
4. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (2000)
5. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-Modelling Based Assembly Techniques for Situational Method Engineering. Information Systems 24 (1999) 209–228
6. Carmel, E., Tjia, P.: Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce. Cambridge University Press, Cambridge, NY (2005)
7. Holmström, H., Fitzgerald, B., Ågerfalk, P. J., Conchúir, E. Ó.: Agile Practices Reduce Distance in Global Software Development. Information Systems Management 23 (to appear)
8. Kirscher, M., Jain, P., Corsaro, A., Levine, D.: Distributed Extreme Programming. In: Proc. International Conference on eXtreme Programming and Flexible Processes in Software Engineering (2001)
9. Lings, B., Lundell, B.: Method-in-Action and Method-in-Tool: Some Implications for Case. In: Proc. 6th International Conference on Enterprise Information Systems (ICEIS 2004) (2004) 623–628
10. Mylopoulos, J., Chung, L., Liao, S., Wang, H., Yu, E.: Exploring Alternatives During Requirements Analysis. IEEE Software 18 (2001) 92–96
11. Rossi, M., Ramesh, B., Lyytinen, K., Tolvanen, J.-P.: Managing Evolutionary Method Engineering by Method Rationale. Journal of the Association for Information Systems 5 (2004) 356–391
12. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice-Hall, Upper Saddle River, NJ (2002)
13. Stolterman, E., Russo, N. L.: The Paradox of Information Systems Methods: Public and Private Rationality. In: Proc. British Computer Society 5th Annual Conference on Methodologies (1997)
14. Weber, M.: Economy and Society. University of California Press, Berkeley, CA (1978)