New Directions on Agile Methods: A Comparative Analysis

Pekka Abrahamsson[a], Juhani Warsta[b], Mikko T. Siponen[b] and Jussi Ronkainen[a]
[a]*Technical Research Centre of Finland, VTT Electronics*
P.O.Box 1100, FIN-90571 Oulu, Finland
[b]*Department of Information Processing Science*
P.O.Box 3000, FIN-90014 University of Oulu, Finland
{Pekka.Abrahamsson, Jussi.Ronkainen}@vtt.fi, {Juhani.Warsta, Mikko.T.Siponen}@oulu.fi

Abstract

Agile software development methods have caught the attention of software engineers and researchers worldwide. Scientific research is yet scarce. This paper reports results from a study, which aims to organize, analyze and make sense out of the dispersed field of agile software development methods. The comparative analysis is performed using the method's life-cycle coverage, project management support, type of practical guidance, fitness-for-use and empirical evidence as the analytical lenses. The results show that agile software development methods, without rationalization, cover certain/different phases of the software development life-cycle and most of them do not offer adequate support for project management. Yet, many methods still attempt to strive for universal solutions (as opposed to situation appropriate) and the empirical evidence is still very limited. Based on the results, new directions are suggested. In principal, it is suggested to place emphasis on methodological quality – not method quantity.

1. Introduction

Agile – denoting "the quality of being agile; readiness for motion; nimbleness, activity, dexterity in motion" (http://dictionary.oed.com) – software development methods attempt to offer once again an answer to the eager business community asking for lighter weight along with faster and nimbler software development processes. This is especially the case with the rapidly growing and volatile Internet software industry as well as with the emerging mobile application environment. The new agile methods have evoked a substantial amount of literature and debates [e.g., 1, 2, 3].

Agile proponents claim that the focal aspects of light and agile methods are simplicity and speed [4-6]. In development work, accordingly, development groups concentrate only on the functions needed immediately, delivering them fast, collecting feedback and reacting rapidly to business and technology changes [7-9].

The emerging of numerous different agile methods has exploded during the last years and is not showing any signs of ceasing. This has resulted in a situation where researchers and practitioners are not aware of the existing approaches or their suitability for varying real-life software development situations. As for researchers and method developers, the lack of unifying research hinders the ability to establish a reliable and cumulative research tradition.

The aim of this paper is to organize, analyze and make sense out of the dispersed field of agile software development methods. Based on the result of the analysis, practitioners are in a better position to understand the various properties of each method and make their judgment in a more informed way. For these purposes, an analytic framework is constructed, which guides the analysis of the different existing methods. To scrutinize the agile methods the following five perspectives are chosen: software development life-cycle including the process aspect, project management, abstract principles vs. concrete guidance, universally predefined vs. situation appropriate (i.e., fit-for-use), and empirical evidence.

The rest of the paper is composed as follows. The second section presents a short overview of the existing agile methods. The third section presents the lenses for the subsequent analysis. The fourth section presents a comparative analysis of the referred methods and the fifth section discusses the significance and the limitations of the findings. The sixth section concludes this study, recapitulating the key findings.

2. An overview of agile methods

In this section the existing agile methods are identified and their objectives briefly introduced. Figure 1 shows the manifold agile software development methods and their interrelationships together with their evolutionary paths. Figure 1, purposefully, extends the considerations beyond the scope of this paper. This means that some, more philosophical meta-level expositions, which have in their turn impinged upon the preceding agile methods either directly or indirectly are included in the diagram. Figure 1 also depicts (i.e., using a dashed line) which methods (or method developers) contributed to the publication of the agile manifesto. (http://www.agilemanifesto.org).

For the purposes of this paper, agile software development in general is characterized by the following attributes: incremental, cooperative, straightforward, and adaptive [10]. Incremental refers to small software releases, with rapid development cycles. Cooperative refers to a close customer and developer interaction. Straightforward implies that the method itself is easy to learn and to modify and that it is sufficiently documented. Finally, adaptive refers to the ability to make and react to last moment changes. Due to limited space, readers are referred to [10] for further discussion on other possible characterizations. In the following, each method's objectives are briefly introduced.

Adaptive software development. Adaptive software development (ASD) [11] attempts to bring about a new way of seeing the software development in an organization, promoting an adaptive paradigm. It offers solutions for the development of large and complex systems. The method encourages incremental and iterative development, with constant prototyping. One ancestor of ASD is "RADical Software Development" [12]. ASD claims to provide a framework with enough guidance to prevent projects from falling into chaos, but not too much, which could suppress emergence and creativity.

Agile modeling. Agile modeling (AM) [13] is a new approach for performing modeling activities. It attempts to adapt modeling practices using an agile philosophy as its backbone. The key focus in AM, therefore, is on modeling practices and cultural principles. The underlying idea is to encourage developers to produce sufficiently advanced models to support acute design needs and documentation purposes. The aim is to keep the amount of models and documentation as low as possible. Cultural issues are addressed by depicting ways to encourage communication, and to organize team structures and ways of working.

Crystal family. The Crystal family of methodologies [14-16] includes a number of different methods from which to select the most suitable one for each individual project. Besides the methods, the Crystal approach also includes principles for tailoring these methods to fit the varying circumstances of different projects. Each member of the Crystal family is marked with a color indicating the 'heaviness' of the method. Crystal suggests choosing the appropriate-colored method for a project based on its size and criticality. Larger projects are likely to ask for more coordination and heavier methods than smaller ones. Crystal methods are open for any development practices, tools or work products, thus allowing the integration of, for example, XP and Scrum practices.

Dynamic systems development method. Dynamic systems development method (DSDM) [17, 18] is a method developed by a dedicated consortium in the UK. The first release of the method was in 1994. The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality, it is preferred to fix time and resources, and then adjust the amount of functionality accordingly. The origins of DSDM are in rapid application development. DSDM can be seen as the first truly agile software development method.

Extreme programming. Extreme programming (XP) [5, 19, 20] is a collection of well-known software engineering practices. XP aims at enabling successful software development despite vague or constantly changing software requirements. The novelty of XP is based on the way the individual practices are collected and lined up to function with each other. Some of the main characteristics of XP are short iterations with small releases and rapid feedback, close customer participation, constant communication and coordination, continuous refactoring, continuous integration and testing, collective code ownership, and pair programming.

Feature-driven development. Feature-driven development (FDD) [21, 22] is a process-oriented software development method for developing business critical systems. The FDD approach focuses on the design and building phases. The FDD approach embodies iterative development with the practices believed to be effective in industry. The specific blend of these ingredients makes the FDD processes unique for each case. It emphasizes quality aspects throughout the process and includes frequent and tangible deliveries, along with accurate monitoring of the progress of the project.

Internet-speed development. Internet-speed development (ISD) [23-25] is arguably the least known approach to agile software development. ISD refers to a situation where software needs to be released fast, thereby requiring short development cycles. ISD puts forth a descriptive, management-oriented framework for addressing the problem of handling fast releases. This framework consists of time drivers, quality dependencies and process adjustments.

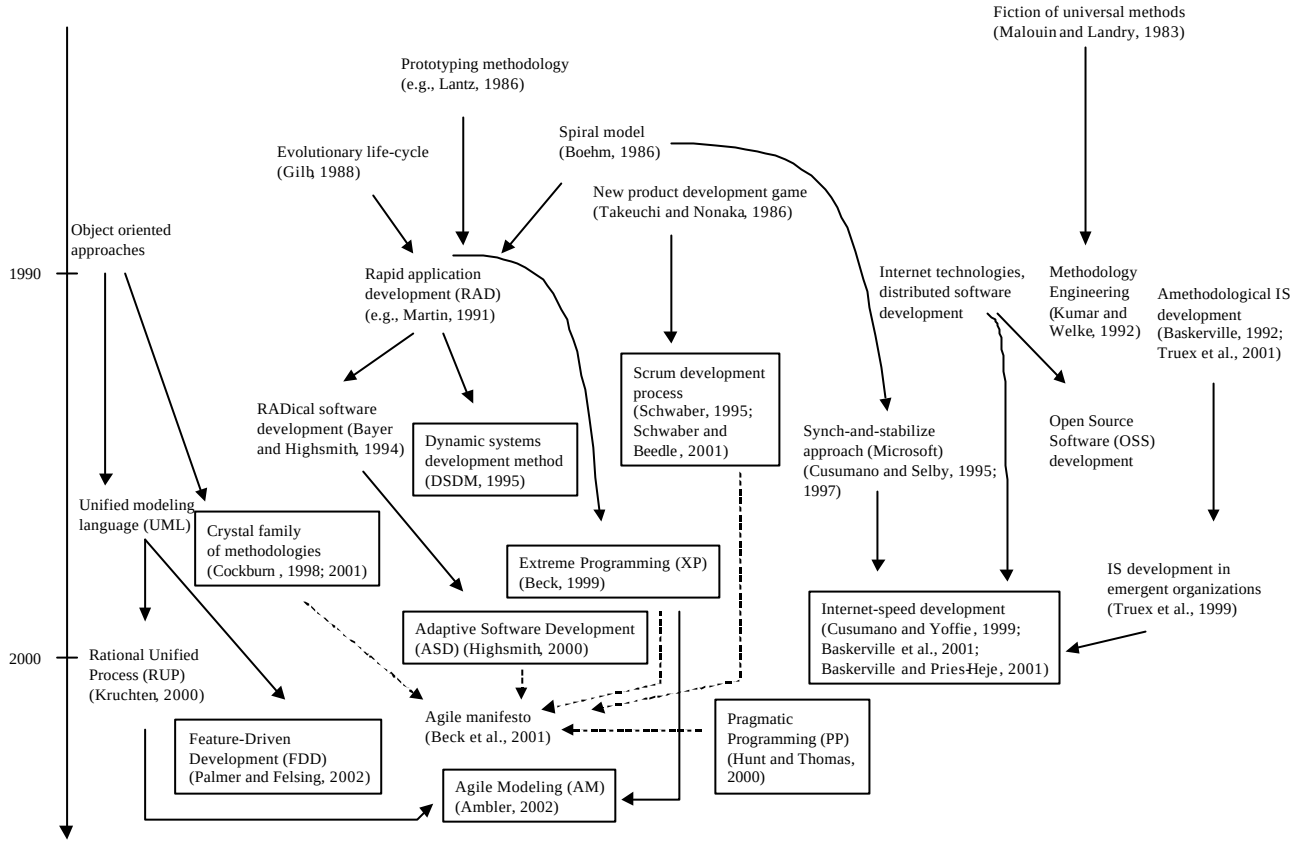

Figure 1. Evolutionary map of agile methods.

These are rules under which companies have survived in ISD. In this context the 'process adjustment' means focusing on good people instead of process, i.e., "if people are mature and talented, there is less need for process" [23, p. 56]. The framework for ISD is considered as more management and business-oriented than other related approaches. ISD draws from the "*Synch-and-stabilize*" approach by Microsoft, aimed at coping with a fast-moving, or even chaotic, software development business [26], and from emergent organizations, which are organizations having a fast pace of organizational change – an opposite to stable organizations [27]. ISD's theoretical background stems from Amethodological IS development [28, 29], which argues that software development is a collection of random, opportunistic processes driven by accident. These processes are simultaneous, overlapping and there are gaps and the development itself occurs in completely unique and idiographic forms. Finally, the development is negotiated, compromised and capricious as opposed to predefined, planned and mutually agreed.

Pragmatic programming. Pragmatic programming (PP) [30] introduces a set of programming "best practices". It puts forward techniques that concretely augment the practices discussed in the other agile methods. PP covers most programming practicalities. The "method" itself is a collection of short tips that focus on day-to-day problems; there are a total of 70 of them. These practices take a pragmatic perspective and place focus on incremental, iterative development, rigorous testing, and user-centered design.

Scrum. The Scrum [31, 32] approach has been developed for managing the software development process in a volatile environment. It is an empirical approach based on flexibility, adaptability and productivity. Scrum leaves open for the developers to choose the specific software development techniques, methods, and practices for the implementation process. It involves frequent management activities aiming at consistently identifying any deficiencies or impediments in the development process as well as the in the practices that are used.

3. Lenses for the analysis

In order to make sense and scrutinize the existing agile methods, proper analytic tools are need. Many such analytical tools have been proposed and used [e.g., 33, 34]. The following five analytical lenses were seen as

relevant and complementary for addressing the research purposes of the paper (Table 1).

Table 1. Lenses for the analysis.

Perspective	Description	Key references
Software development life-cycle	Which stages of the software development life-cycle does the method cover	[35, 36]
Project management	Does the method support project management activities	[37, 38]
Abstract principles vs. concrete guidance	Does the method mainly rely on abstract principles or does it provide concrete guidance	[39]
Universally predefined vs. situation appropriate	Is the method argued to fit *per se* in all agile development situations	[37]
Empirical evidence	Does the method have empirical support for its claims	[37, 40-42]

A *software development life-cycle* is a sequence of processes that an organization employs to conceive, design, and commercialize a software product [35, 36]. A software development life-cycle perspective is needed to observe which phases of the software development process the agile methods cover. A software development life-cycle can be seen as consisting of nine phases [e.g., 43]: project inception, requirements specification, design, coding, unit test, integration test, system test, acceptance test, system in use (i.e., maintenance). Life-cycle coverage also includes that the process through which the software production proceeds needs to be identified.

Methods should be efficient (as opposed to time and resource consuming) [37]. Efficiency requires the existence of *project management* activities to enable the proper execution of software development tasks. Project management is thus a support function that provides the backbone for efficient software development [38]. Therefore, a project management perspective can be seen as a relevant dimension in the evaluation of agile software development methods.

Software development methods are often used for other purposes than originally intended by their authors [39]. For example, Wastel [44] observed that methodologies act as a social defense operating as a set of organizational rituals. Thus, in order to evaluate whether the methods can be used for other purposes than the necessary fiction of control [39], a perspective of *abstract principles versus concrete guidance* is needed. It will be evaluated whether the agile software methods provide any concrete guidance or do they rely on abstract rules of thumb. "Respect people" without instructing how to actually perform it is an example of an abstract principle. Concrete guidance, on the other hand, refers to practices, activities and work products at the different phases of the software development life-cycle.

The *Universally predefined versus situation appropriate* viewpoint stems from the works of Kumar and Welke [37], Malouin and Landry [45], and Truex *et al*.[29]. Universally predefined means a view according to which there is one method or methodology that fits as such to all (agile) development situations. In other words, a universally predefined viewpoint proposes one ready-made solution to all (agile) SW development endeavors. The view of situation appropriate, as opposed to universally predefined, holds that there can be no predetermined set of methods that would fit *per se* to all (agile) software development situations. In other words, situation appropriate refers to the extent to which a method is adjustable depending on the situation.

Empirical support is needed to see what kind of empirical evidence the agile methods are grounded upon. Clearly, one of the most important aspects of research is the use of proper research methods [46]. Since software development has a strong practical orientation, there is need for studying the different agile methods in real life situations [cf., 40, 41, 42], i.e., using empirical studies. This viewpoint explores what, if any, empirical support the different methods have.

4. Comparative analysis of the existing agile methods

In this section, the existing agile methods are compared using the analytical lenses defined in section three. Thus, for each method the software life-cycle coverage, project management support, type of guidance, situation appropriateness and the level of empirical support is evaluated. Each perspective will be analyzed separately.

Figure 2 serves for the purposes of the first three lenses. Each method is divided into three bars. The uppermost bar indicates whether a method provides support for project management (analyzed in section 4.1). The middle bar indicates whether a process through which the software production proceeds is described (pertaining to software development life-cycle analysis). The length of the bar shows which phases of software development are supported by different agile methods (analyzed in section 4.2). Finally, the lowest bar shows whether a method relies mainly on abstract principles (white color) or does it provide concrete guidance (gray color). This will be analyzed in section 4.3.

In general, a gray color in a block indicates that the method covers the perspective analyzed while a white color indicates lack of such support.

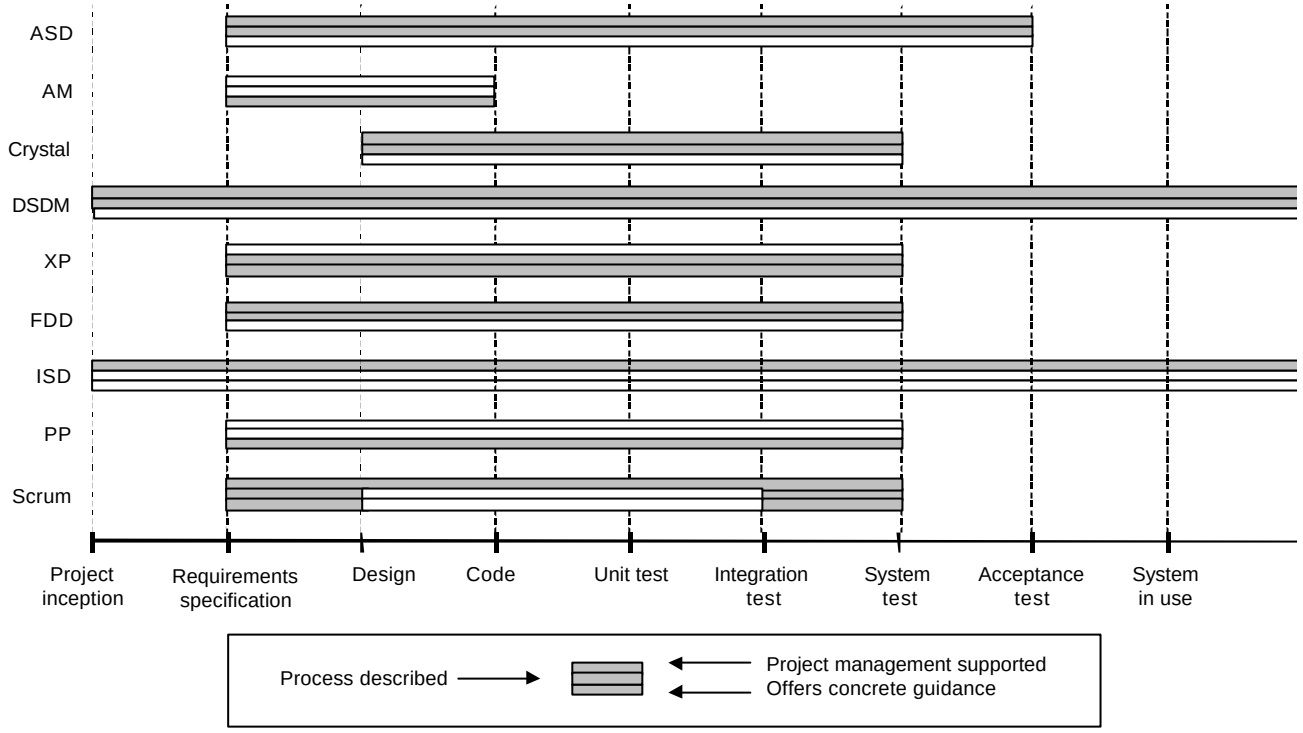

Figure 2. Comparing life-cycle, project management and concrete guidance support.

4.1. Project management

Agile software development methods differ to a large degree in the way they cover project management (uppermost bar in Figure 2). Currently, AM and PP do not address the managerial perspective. XP has recently been supplemented with some guidelines on project management [47], but it still does not offer a comprehensive project management view. Scrum, on the other hand, is explicitly intended for the purpose of managing agile software development projects. Thus, Schwaber and Beedle [32] suggest the use of other methods to complement a Scrum based software development approach, naming XP as one alternative.

The approach promoted by ASD is the *adaptive (leadership-collaboration) management model* [11]. The point of view in ASD is on changing the software development culture, essentially stating that management will also have to bend in response to changes in projects. FDD offers means for planning projects by product features, and tracking the projects' progress. FDD also believes in empowering project managers; within an FDD project, the project manager has the ultimate say on project scope, schedule, and staffing [21].

DSDM suggests a framework of controls to supplement the rapid application development approach. All of these controls are designated to increase organizational ability to react to business changes [18], which has become commonplace nowadays in all agile software development approaches. Therefore, the DSDM approach towards project management is largely about facilitating the work of the development teams, with daily tracking of the project's progress. Crystal's solution to project management focuses on increasing the ability to choose the correct method for the purpose [16].

4.2. Software development life-cycle

Figure 2 shows that different agile methods are focused on different aspects of the software development life-cycle. DSDM is an independent method in the sense that it attempts to provide complete support over all life-cycle phases. The Internet-speed development approach also addresses all the phases of the software development life-cycle but only at a managerial level. Others are more focused. ASD covers all other phases expect for project inception, acceptance test and system in use. AM aims at providing modeling support for requirements specification and design phases.

The Crystal family covers the phases from design to integration test. XP, PP, FDD and Scrum are focused on requirements specification, design, implementation (except for Scrum) and testing up until the system test.

From the process perspective, AM, ISD, Scrum (for the implementation part) and PP approaches do not emphasize (or have not described) the process through which the software development proceeds. AM and PP are supplements to other methods. Thus, in the case of AM and PP the lack of a process perspective seems reasonable. However, ISD lacks clarity in this regard. In the other agile software development methods (i.e., ASD, Crystal, DSDM, XP, FDD and Scrum), the development process has been described.

4.3. Abstract principles vs. concrete guidance

The lowest bar in Figure 2 indicates whether the method relies on concrete guidance (gray color) or on abstract principles (white color).

A method that is useful and efficient should not refer to abstract principles only in forming the core of the method [39]. It is claimed that abstract principles are useless if not supported with concrete guidance [48]. Concrete guidance, in this context, refers to practices, activities and work products that characterize and provide guidance on how a specific task can be executed. For example, FDD lays down eight practices that "must" be used if compliance with the FDD development rules is to be valued [21]. They continue that the "team is allowed to adapt them according to their experience level". Especially, in the case of these "must" practices, concrete guidance should be provided how, in practice, this adaptation can be executed. If such guidance is missing (as it is, in this case), we interpret it as a reliance to abstract principles.

Based on this distinction, it was found that five out of nine agile software development methods included in the analysis place emphasis on abstract principles over concrete guidance. ASD is more about concepts and culture than software practice. Crystal, depending on the system criticality and project size, mandates certain practices but does not provide any guidance on how to execute them. DSDM states that due to the fact that each organization is different no practices are detailed [18]. Instead, organizations should develop the practices themselves. Concrete guidance on how this should be done, is not provided. Internet-speed development establishes certain practices or principles that should be in place. However, it does not offer any concrete guidance on how one should actually carry out the ideas of e.g. "always analysis" or "dynamic requirements negotiation".

AM, XP and PP have been directly derived from practical settings. Their purpose and goal is to feed collected "best practices" back into the actual practice of software development. Hence, these three methods are strong in their focus on concrete guidance. Whether the guidance they offer, however, is of value and correct falls beyond the scope of this paper (whether the guidance is supported by empirical evidence is, however, analyzed in section 4.5). Scrum defines the practices and offers guidance for the requirements specification phase as well as the integration testing phase. Implementation phases, as stated earlier, are not a part of the method.

4.4. Universally predefined vs. situation appropriate

The goal of agile software development is to increase the ability to react and respond to changing business, customer and technological needs at all organizational levels. Agile software development methods are used in a hope of nearing toward this goal. While agility refers to nimbleness, a method aiding this process needs to be nimble as well. In other words, in order to increase the level of agility, the tools (i.e., agile software development methods) used in the development process need to be agile also, i.e. situation appropriate. Situation appropriate therefore means that a method can be adjusted to different situations. A method should be flexible enough to enable adjustments to be made on-the-fly, i.e. whenever the situation calls for an adjustment. A universally predefined viewpoint proposes one ready-made solution to all agile software development endeavors.

Table 2 presents the results of the analysis regarding situation appropriateness and empirical evidence. The results are shown here using the following scale:

+ Situation appropriate, adjustable
– Universally predefined solutions, not adjustable

Empirical evidence will be discussed in the next subsection. Here, the following scale is used:

+ Some empirical evidence exists
– No empirical evidence exists

Table 2. Situation appropriateness and empirical evidence

Perspective:	Situation appropriateness	Empirical evidence
ASD	+	-
AM	+	-
Crystal	-	-
DSDM	+	-
XP	+	+
FDD	-	-
Internet speed	+	+
PP	+	-
SCRUM	+	+

Cockburn's [16] Crystal family of methodologies explicitly provides criteria on how to select the methodology for a project. The selection is made based on project size, criticality and priority [49]. Based on these factors a decision is made about which methodology should be used. However, when the project is underway the situation appropriateness is diminished. This interpretation is based on the fact that Crystal methods offer prescriptive guidance. This means that Crystal e.g. enforces certain rules such as "the projects always use incremental development cycles with a maximum increment length of four months" Thus, the adjustment is made by choosing one of the several "universal solutions" such as Crystal Clear or other.

The analysis shows further that FDD also has goals that are grounded on imperatives or prescriptive guidance similar to Crystal. Palmer and Felsing [21, p. 35] explain that FDD is "built around a core set of 'best practices'." All of these practices must be used in order to "get the full benefit that occurs by using the whole FDD process". FDD is claimed to suit for "any software development organization that needs to deliver quality, business-critical software systems on time." [21, p. xxiii]. Thus, FDD, Crystal and DSDM all represent universal prescriptions that claim to have the suitability for all agile software development situations, scopes and projects.

The DSDM Consortium [17] has published a method suitability filter in which three areas are covered: business, systems and technical [18]. The filter involves a series of questions such as "Are the requirements flexible and only specified at a high level?" or "Is functionality going to be reasonably visible at the user interface?" and some rationalization about which type of an answer would yield greater benefits if DSDM were to be applied. While the method filter is predominantly about deciding whether the method itself is applicable or not, a recent study [50] showed how the DSDM was tailored in a CMM context to a project's purposes, implying the necessary situation appropriate characteristics.

Further, the ASD, AM, XP, ISD, PP and Scrum approaches allow situation appropriate modifications. For example, regarding XP Beck [5, p.77] suggests: "If you want to try XP, … don't try to swallow it all at once. Pick the worst problem in your current process and try solving it the XP way." Beck maintains is that there is no process that fits every project as such, but rather the practices should be tailored to suit the needs of individual projects. We interpret this in such a way that the number of adjustments is not limited. One of XP's practices, namely "just rules", implies that while the rules are followed, they can be changed if a mutual understanding among the development team is achieved. This implies that XP supports situation appropriateness.

AM and PP offer supplemental practices and concrete guidance on how and when to apply them in actual software development work. Their authors describe the situations and rationale for applying the practices suggested but refrain from offering prescriptive guidance.

4.5. Empirical evidence

This viewpoint explores what, if any, empirical support the different agile software development methods have. Table 2 presents the results of the analysis regarding the empirical evidence (right column).

While ASD, Crystal, FDD and Scrum are derived from subjective practical experience, the way they were developed is not based on reliable and systematic research. Thus their solutions lack real empirical support. AM and PP do not offer empirical support for their suggestions, either. Regarding Scrum, Schwaber and Beedle [32, p.31] claim e.g., that "*[Scrum] practices have been established through thousands of Scrum projects*". None of these projects are cited however, which invites skepticism regarding the validity of the empirical evidence. More recently, however, a study [51] can be found where Scrum was tested in real-life projects, thus showing that the body of needed industrial evidence is gradually growing, and that Scrum has some empirical support.

Internet-speed development has a degree of empirical support. It is based on qualitative case studies in nine companies [23]. We see that these cases are aimed at increasing our understanding of how Interned-speed companies survived, not proposing "laws" for making successful agile SW development.

XP is not based on systematic research but rather on the expertise and experiences of a few individual software engineers. However, parts of XP have been studied empirically. For example, empirical studies exist on pair programming [52, 53]. Maurer and Martel [54] include some concrete numbers regarding the productivity gains using XP in a web development project. Furthermore, there are some experience reports available of applying XP in industrial [e.g., 55-57] and university [e.g., 9, 58] settings. These studies provide the necessary insight on the possibilities and restrictions of XP.

DSDM has been developed by a dedicated consortium. Stapleton (1997) claims that empirical evidence exists in the form of experience reports (i.e., white papers) that are shared with the members of the consortium. However, since they are not made publicly available we interpret that the claims of DSDM are not empirically supported.

5. Discussion

In this section, the results of the comparative analysis are discussed. The purpose is to identify the principal implications for research and practice in the field of

software engineering. Table 3 summarizes these implications.

Table 3. Results and implications of the study

Perspective	Description of the results	Implications
Software development life-cycle	Methods, without rationalization, cover different phases of the life-cycle.	Life-cycle coverage needs to be explained and interfaces with phases not covered need to be clarified.
Project management	While most methods appear to cover project management, true support is missing.	Conceptual harmonization is needed. Project management can not be neglected.
Abstract principles vs. concrete guidance	Abstract principles dominate the method literature and developers' minds.	Emphasis should be placed on enabling practitioners to utilize the suggestions made.
Universally predefined vs. situation appropriate	Universal solutions dominate the literature.	More work on how to adopt agile methods in different development situations is needed.
Empirical support	Empirical evidence is limited; most of the research is at conceptual level	More empirical, situation-specific, experimental work is needed; results need to be publicly available.

Software development life-cycle: Different agile methods cover different phases of the software development life-cycle. The rationalization of phases covered was, however, missing. The question that must, therefore, be raised is whether it is more profitable to cover more and to be more extensive, or cover less and to be more precise. Completeness, a notion introduced by Kumar and Welke [37], requires that a method is complete as opposed to partial. In the analysis it was realized that "completeness" is an element that must be associated both with vertical (i.e., level of detail) and horizontal (i.e., life-cycle coverage) dimensions. None of the methods evaluated were either extensive or precise. The practitioners, currently, have partial solutions to problems that cover a wider area than the methods do. It is suggested that method developers concentrate more on specialization than generalization in the areas of their expertise. On one hand, methods that cover too much ground, i.e. all organizations, phases and situations, are too general or shallow to be used. On the other hand, methods that cover too little (e.g., one phase) may be too restricted or lack a connection to other methods.

Project management: Software engineering is a practically oriented field. Yet, while most (i.e., five out of nine) agile methods do incorporate support for project management, true support is scarce. Considering this perspective from a method feasibility point of view, efficient project management is of utmost importance when agile principles such as daily builds, short release-cycles, etc. are followed. Moreover, and more importantly, the concepts of, e.g., release and daily builds differ from one method to another. This invites more confusion than clarity. It appears that the method developers are aiming for their niche by using purposefully differing terminology. Practitioners, especially project managers, are in a difficult position when a decision of the most suitable approach should be made. The operational success of any method lies on its ability to be incorporated in the software project's daily rhythm. We, thus, maintain that project management considerations need to be addressed explicitly to ensure the alignment between a developer and the project management.

Abstract principles vs. concrete guidance: Only three out of nine agile methods offer concrete guidance. Abstract principles appear to dominate the agile method literature and developers' minds. The agile community is more concerned about getting acceptance to proposed values than in offering guidance on how to use the operative versions of these values. Currently, concrete guidance exists mostly in methods that are very limited in their scope (i.e., AM) or depth (i.e., PP). More work is needed in determining how the claimed practices, activities and work products are made to operate in different organizations and situations so that practitioners have a solid base on which to constitute their decisions.

Universally predefined vs. situation appropriate: Some of the well-known agile methods (e.g., FDD) were found to be universally predefined. Nevertheless, the few methods (e.g., Crystal) that recognize the fact that "one size does not fit to all situations", still do not expound any guidance on how this fitting process may be done. This being the case, forthcoming methods and studies should pay particular attention to situation appropriateness, and offer guidance on how the methods should be used in different agile software development situations. This also requires the ability to identify the particular situations in which these fitting or adjustment activities need to be done.

Empirical support: Empirical evidence, based on rigorous research, is scarce. Only three out of nine methods have some empirical support for their claims. Yet, it should be noted that some of the methods (e.g., XP) are

increasingly producing more and more empirical studies, which is bound to mature the methodological base. Lack of empirical evidence results in practitioners not having reliable evidence on the real usability and the effectiveness of particular methods. They do not know whether the existing methods really make sense – do the methods describe proven wisdom or folktales. As for researchers and method developers, the lack of empirical evidence does not help in establishing a reliable and cumulative research tradition. Currently, researchers do not have access to reliable evidence on existing works. Such information is, however, necessary in pondering which parts of the previous works have a truth in them, and to decide to which extent future research can be based on existing wisdom. Thus, more empirical work is needed to study the implications of the different agile methods in different organizations and situations. Also, it should be noted that empirical evidence – that is, if it exists – should be publicly accessible. Currently, empirical studies on DSDM are made available only to members of the consortium.

Empirical works that study the effects of particular methods, their ease of use, costs, and possible negative implications for different sizes and lines of business, are needed in particular. The empirical work should use both qualitative and quantitative research methods to study these issues.

6. Conclusions

Agile software development methods have evoked a substantial amount of literature and debates. However, academic research on the subject is still scarce, as most existing publications are written by practitioners or consultants. Yet, many organizations are considering to use or have already applied practices that are claimed to make their way of performing and delivering software more agile.

The aim of this paper is to attempt to make sense out of the jungle of emerged agile software development methods. Based on the result of the analysis, practitioners are in a better position to understand the various properties of each method and make their judgment in a more informed way. The approach chosen for the purpose was that of comparative analysis. The analytical lenses included five perspectives: software development life-cycle including the process aspect, project management, abstract principles vs. concrete guidance, universally predefined vs. situation appropriate, and empirical evidence.

We observed that methods, without rationalization, cover certain/different phases of the life-cycle. A majority of them did not provide true support for project management, and abstract principles appeared to dominate the current method literature and developers' minds. While universal solutions have a strong support in the respective literature, empirical evidence is currently very limited.

Based on the above, new directions were offered. Namely, it was suggested that emerging new agile methods need to clarify their range of applicability and explain the interfaces to those parts of the software development life-cycle which are not a part of the chosen focus. In addition it was suggested that emphasis should rather be placed on method specialization than generalization. However, this specialization should enhance the conceptual harmonization rather than to work against it. Furthermore, the project management perspective cannot be neglected, if a method is to encroach on day-to-day software development practices. Emphasis should also be placed on enabling practitioners to utilize the suggestions made. This requires placing the focus in method development on empirically validated situation-specific solutions.

The current trend on agile methods has focused on fabricating a pile of conceptual methods. Instead of hurrying to introduce yet more agile methods, the method developers should pay particular attention to address the problems described. The field is crying for sound methods, i.e. methodological quality - not method quantity.

7. References

[1] J. Highsmith, "The great methodologies debate: Part 2," *Cutter IT Journal*, vol. 15, 2002.

[2] J. Highsmith, "The great methodologies debate: Part 1," *Cutter IT Journal*, vol. 14, 2001.

[3] E. Yourdon, "Light methodologies," *Cutter IT Journal*, vol. 13, 2000.

[4] R. McCauley, "Agile Development Methods Poised to Upset Status Quo," *SIGCSE Bulletin*, vol. 33, pp. 14 - 15, 2001.

[5] K. Beck, "Embracing Change With Extreme Programming," *IEEE Computer*, vol. 32, pp. 70-77, 1999.

[6] J. Highsmith and A. Cockburn, "Agile Software Development: The Business of Innovation," *Computer*, vol. 34, pp. 120-122, 2001.

[7] B. Boehm, "Get Ready For The Agile Methods, With Care," *Computer*, vol. 35, pp. 64-69, 2002.

[8] M. Fowler and J. Highsmith, "Agile methodologists agree on something," *Software Development*, vol. 9, pp. 28-32, 2001.

[9] M. M. Müller and W. F. Tichy, "Case Study: Extreme Programming in a University Environment,"

presented at 23rd International Conference on Software Engineering, Toronto, 2001.

[10] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods: Review and Analysis*. Espoo, Finland: Technical Research Centre of Finland, VTT Publications 478, Available online: http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf, 2002.

[11] J. A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, NY: Dorset House Publishing, 2000.

[12] S. Bayer and J. Highsmith, "RADical software development," *American Programmer*, vol. 7, pp. 35-42, 1994.

[13] S. Ambler, *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. New York: John Wiley & Sons, Inc. New York, 2002.

[14] A. Cockburn, *Surviving Object-Oriented Projects: A Manager's Guide*, vol. 5: Addison Wesley Longman, 1998.

[15] A. Cockburn, *Writing Effective Use Cases, The Crystal Collection for Software Professionals*: Addison-Wesley Professional, 2000.

[16] A. Cockburn, *Agile Software Development*. Boston: Addison-Wesley, 2002.

[17] DSDMConsortium, *Dynamic Systems Development Method, version 3*. Ashford, Eng.: DSDM Consortium, 1997.

[18] J. Stapleton, *Dynamic systems development method - The method in practice*: Addison Wesley, 1997.

[19] K. Beck, *Extreme programming explained*. Reading, Mass.: Addison-Wesley, 1999.

[20] K. Beck, *Extreme Programming Explained: Embrace Change*, 2000.

[21] S. R. Palmer and J. M. Felsing, *A Practical Guide to Feature-Driven Development*, 2002.

[22] P. Coad, E. LeFebvre, and J. De Luca, *Java Modeling In Color With UML: Enterprise Components and Process*: Prentice Hall, 2000.

[23] R. Baskerville, L. Levine, J. Pries-Heje, B. Ramesh, and S. Slaughter, "How Internet companies negotiate quality," *IEEE Computer*, vol. 34, pp. 51-57, 2001.

[24] R. Baskerville and J. Pries-Heje, "Racing the E-bomb: How the Internet is redefining information systems development methodology," in *Realigning research and practice in IS development*, B. Fitzgerald, N. Russo, and J. DeGross, Eds. New York: Kluwer, 2001, pp. 49-68.

[25] M. A. Cusumano and D. B. Yoffie, "Software development on Internet time," *IEEE Computer*, vol. 32, pp. 60-69, 1999.

[26] M. A. Cusumano and R. W. Selby, "How Microsoft builds software," *Communications of the ACM*, vol. 40, pp. 53-61, 1997.

[27] D. P. Truex, R. Baskerville, and H. Klein, "Growing systems in emergent organizations," *Communications of the ACM*, vol. 42, pp. 117-123, 1999.

[28] R. Baskerville, J. Travis, and D. P. Truex, "Systems without method: The impact of new technologies on information systems development projects," in *Transactions on the impact of computer supported technologies in information systems development*, K. E. Kendall, K. Lyytinen, and J. I. DeGross, Eds. Amsterdam: Elsevier Science Publications, 1992, pp. 241-260.

[29] D. P. Truex, R. Baskerville, and J. Travis, "Amethodological systems development: The deferred meaning of systems development methods," *Accounting, Management and Information Technology*, vol. 10, pp. 53-79, 2001.

[30] A. Hunt, Thomas, D., *The Pragmatic Programmer*: Addison Wesley, 2000.

[31] K. Schwaber, "Scrum Development Process," presented at OOPSLA'95 Workshop on Business Object Design and Implementation, 1995.

[32] K. Schwaber and M. Beedle, *Agile Software Development With Scrum.* Upper Saddle River, NJ: Prentice-Hall, 2002.

[33] J. Iivari and R. Hirscheim, "Analyzing information systems development: A comparison and analysis of eight IS development approaches," *Information Systems*, vol. 21, pp. 551-575, 1996.

[34] T. W. Olle, H. G. Sol, and A. Verrijn-Stuart, *Information systems design methodologies: A comparative review*. Amsterdam: North-Holland, 1982.

[35] B. W. Boehm, "A spiral model of sofware development and enhancement," *IEEE Computer*, vol. 21, pp. 61-72, 1988.

[36] G. Cugola and C. Ghezzi, "Software Processes: a Retrospective and a Path to the Future," *Software Process Improvement and Practice*, vol. 4, pp. 101-123, 1998.

[37] K. Kumar and R. J. Welke, "Methodology engineering: A proposal for situation-specific methodology construction," in *Challenges and strategies for research in systems development*, W. W. Cotterman and J. A. Senn, Eds. New York: John Wiley & Sons, 1992, pp. 257-269.

[38] T. Gilb, *Principles of Software Engineering Management*. Wokingham, UK: Addison-Wesley, 1988.

[39] J. Nandhakumar and J. Avison, "The fiction of methodological development: a field study of

information systems development," *Information Technology & People*, vol. 12, pp. 176-191, 1999.

[40] V. R. Basili and F. Lanubile, "Building knowledge through families of experiments," *IEEE Transactions on Software Engineering*, vol. 25, pp. 456-473, 1999.

[41] V. R. Basili, "The role of experimentation in software engineering: Past, present and future," presented at Keynote address in 18th International Conference on Software Engineering (ICSE18), Berlin, Germany, 1996.

[42] N. Fenton, "Viewpoint Article: Conducting and presenting empirical software engineering," *Empirical Software Engineering*, vol. 6, pp. 195-200, 2001.

[43] I. Sommerville, *Software engineering*, Fifth ed. New York: Addison-Wesley, 1996.

[44] D. G. Wastell, "The fetish of technique: methodology as a social defence," *Information Systems Journal*, vol. 6, pp. 25-49, 1996.

[45] J. L. Malouin and M. Landry, "The miracle of universal methods in systems design," *Journal of Applied Systems Analysis*, vol. 10, pp. 47-62, 1983.

[46] A. F. Chalmers, *What is this thing called Science?*, Third ed. Buckingham: Open University Press, 1999.

[47] R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. Upper Saddle River, NJ: Addison-Wesley, 2001.

[48] D. Fiery, *Secrets of a super hacker*. Port Townsend, Washington, USA: Loompanics Unlimited, 1994.

[49] A. Cockburn, "Selecting a project's methodology," *IEEE Software*, vol. 17, pp. 64-71, 2000.

[50] M. N. Aydin and F. Harmsen, "Making a method work for a project situation in the context of CMM," presented at Product Focused Software Process Improvement (Profes 2002), Rovaniemi, Finland, 2002.

[51] L. Rising and N. S. Janoff, "The Scrum software development process for small teams," *IEEE Software*, vol. 17, pp. 26-32, 2000.

[52] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software*, vol. 17, pp. 19-25, 2000.

[53] J. Haungs, "Pair programming on the C3 project," *Computer*, vol. 34, pp. 118-119, 2001.

[54] F. Maurer and S. Martel, "On the Productivity of Agile Software Practices: An Industrial Case Study," 2002.

[55] J. Grenning, "Extreme Programming and Embedded Software Development," presented at Embedded Systems Conference 2002, Chicago, 2002.

[56] P. Schuh, "Recovery, Redemption, and Extreme Programming," *IEEE Software*, vol. 18, pp. 34-41, 2001.

[57] A. Anderson, R. Beattie, K. Beck, D. Bryant, M. DeArment, *et al.*, "Chrysler Goes to "Extremes". Case Study.," *Distributed Computing*, pp. 24-28, 1998.

[58] J. R. Nawrocki, B. Walter, and A. Wojciechowski, "Comparison of CMM level 2 and eXtreme programming," presented at 7th European Conference on Software Quality, Helsinki, Finland, 2002.