The Impact of Agile Methods on Software Project Management

Michael Coram and Shawn Bohner
Department of Computer Science
Virginia Polytechnical Institute and State University
Blacksburg, Virginia 24061
{mcoram, sbohner}@vt.edu

Abstract

As more and more software projects engage Agile Methods, there are emerging patterns of success and failure. With growing adoption of Agile Methods, project managers increasingly need to understand the applicability to their projects and factors that drive key project performance characteristics. While some organizations affirm that Agile Methods solve all their problems, few have shown consistent success over a range of typical software projects. Agile Methods have advantages, especially in accommodating change due to volatile requirements. However, they also present concomitant risks with managing the many dependent pieces of work distributed across a large project. Use of Agile Methods therefore presents a set of tradeoffs. This paper examines the impact of Agile Methods on the people involved in a project, the process under which a project is developed, and on the project itself in an attempt to allow project managers to evaluate the applicability using an agile method.

1. Introduction

Software engineering, as a discipline, confronts two key challenges that separate it from other engineering disciplines. Software, a conceptual and often intangible product, changes and evolves at a much higher rate than integrated circuits or steel. While software is changeable, there is an increased cost the later in a project lifecycle the change occurs [1]. This is true to a lesser degree in tangible products since measurable tests of the requirements and design can be more readily applied. Recognition of this fact has lead to the emergence of a set of *Agile Methods* that embrace change and manage the related risks [2].

Many such Agile Methods have been introduced over the last decade, including eXtreme Programming (XP) [2], SCRUM [3], and Dynamic System Development Methodology (DSDM) [4]. While these methods differ in their specifics, they share a common goal of enabling teams to more rapidly respond to change. As changes are costly to accommodate later in the project [5], the ability to respond rapidly to change reduces project risks and their costs [2].

While Agile Methods are effective in some contexts, large and complex software products often require systematic discipline with the requisite process overhead to ensure success. The challenge for managers is to determine whether an Agile Method is appropriate for a given set of project activities. Concomitantly, addressing the risks involved with their use warrants attention. All methodologies have risks, and understanding those risks and finding ways to monitor, mitigate, and manage those risks is an important aspect of software project management.

This paper examines the impact of Agile Methods on software project management to illuminate some of the strengths and weaknesses so project managers can make more informed decisions. We present the impact in terms of People, Process, and Project [6].

2. A Brief Look at Agile Methods

The emergence of several Agile Methods over a given span of less than a decade is evidence that the principles that they espouse warrant examination. We briefly present three key Agile Methods to provide a flavor of the principles presented in these different approaches. We then tie them together in discussing their convergence in the Agile Manifesto [7].

2.1 Extreme Programming

Perhaps the most recognizable Agile Method, eXtreme Programming (XP), has the overriding goal "to get the project at hand done." No fan fare, no magic bullets – just apply a series of principles that work. The life cycle of XP consists of five phases: Exploration, Planning, Iterations to Release, Productionizing, Maintenance, and Death. [2]

Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05)
0-7695-2308-0/05 $20.00 © 2005 **IEEE**

The Exploration phase typically takes a few weeks to a few months for customers to provide requirements for the first release. At the same time, the project team becomes familiar with the technology, tools, and practices they will use on the project.

In the Planning phase, the project team spends several days working with the customer to prioritize the capabilities needed for the first release. The developers estimate effort required and the team lead draws up a release schedule not to exceed two months.

The Iterations to Release phase goes through several iterations to produce the first release. Each iteration takes one to four weeks, and at the end of each, the functional tests are executed. Completion of the last iteration marks ready for Productionizing.

In the Productionizing phase, the project team conducts additional performance testing and checking to ensure the release meets the customer requirements. New changes may be introduced here and the decision must be made whether they should be included in the current release. If they are not placed in the current release, they will be recorded for later implementation in subsequent releases. This phase concludes with the release delivered to the customer.

In the Maintenance phase, the team produces new iterations of the software product to implement changes and new feature requests raised in the previous phase. These include corrective, perfective, and adaptive changes incurred during maintenance.

As the software approaches obsolescence, and customers have fewer feasible features to implement, the Death phase entails completing all necessary documentation and the disposition the system is planned. This phase occurs when the value proposition for evolving the system further no longer exists (too expensive to change and low investment value).

2.2 SCRUM

SCRUM [3] development involves several environmental and technical variables that are likely to change during the process. SCRUM concentrates on how teams can be organized to produce software in a constantly changing environment. Modeled after the game of Rugby, the SCRUM life cycle consists of three phases: Pre-game, Development, and Post-game.

In the Pre-game phase, there are two sub-phases: Planning and Architecture/High-level design. Planning entails defining the system based on a Product Backlog List (updated often with features and modifications) which contains all the currently known requirements. These are prioritized and the effort needed is estimated. In the Architecture sub-phase, the design is elaborated and refined based on the backlog list.

In the development phase, iterative cycles of development called "Sprints" are executed to develop new functions and enhance the system. Each Sprint includes: requirements, analysis, design, evolution and delivery. Each Sprint spans from one week to one month. Three to eight Sprints are executed in the development process before the system is completed.

The post-game phase concludes the effort and delivers the release with no additional features or modifications. Unlike XP, there is no specific phase for disposition of the system.

2.3 Dynamic System Development Method

One key aspect that distinguishes the DSDM approach is that it fixes time and resources first and then adjusts the amount of functionality accordingly. This resources-first process consists of five phases: Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, and Implementation. The last three phases are iterative and incremental – restricting iterations within time-boxes (pre-defined periods of time, where the iteration must end within the time-box).

In the Feasibility Study phase, the project is assessed, and the decision on whether or not DSDM is appropriate for the effort. A feasibility report and a development plan are produced over a few weeks.

In the business study phase, key characteristics of the business and technology are assessed culminating in a system architecture definition and an outline prototyping plan. The architecture definition is the initial version of the system definition and it may change as the project proceeds. The prototyping plan outlines the prototyping strategy and the configuration management approach.

During the functional model iteration phase, the project evolves through functional iterations where each iteration involves some enhancements and the increments are directed toward the final system. This phase entails four products that reflect the process: prioritized list of functions, functional prototype(s) review documents, non-functional requirements and risk analysis of further development.

The design and build iteration produces the system that meets the minimum set of requirements and iterate the system based on the customer's comments. Systematically, through a series of iterations and increments, the software is elaborated and refined in a consumable form for the customer to review.

In the implementation phase the system is formally transferred to the actual product. The system is delivered to the customer and any subsequent increments are planned.

2.4 Agile Manifesto Ties It Together

The "Agile Manifesto" provides a good overview of the intent of Agile Methods [7]. The following values express the tenor of the principles employed:

- individuals and interactions over process and tools
- working code over comprehensive documentation
- customer collaboration over contract negotiation
- responding to change over following a plan

In each of these values, the Manifesto is not saying that the second item is not important, just that it is less important than the first item.

Software is inherently challenging because of its constant change. Processes and tools cannot accommodate all of these changes, so people have to pick up the slack. Valuing people over process allows for more creativity in solutions. It implies that even the best process cannot compensate for the shortcomings of individuals [9].

Documentation, while valuable, takes time to write and maintain. However, it is less valuable than a working product. While some Agile Methods promote prototyping (e.g. Adaptive Software Development [9]), others encourage building simple, but completely functional products quickly as possible (e.g. XP [2]).

Customer involvement is promoted in all agile methods. A customer representative is expected to be available and to be "committed, knowledgeable, collaborative, representative, and empowered" [11]. Allowing the customer to use the product quickly is a form of customer collaboration. It also allows the customer to change his or her mind. Thus, instead of writing contracts, which would then need to be change, the customer instead is encouraged to actively participate in the development effort.

Finally, responding to change is considered more important than dogmatically following a plan because a plan is only as good as when it was initially written. If things change, then the plan must as well. But, changes can often happen faster than the plan can be modified. This is not to imply that Agile Methods assume a "hacking mentality" where code is just written without consideration of any plan. Instead, it means that any plan must be lightweight and easily modifiable. The "plan" might simply be a set of post-it notes on a whiteboard (as is used in SCRUM [12]).

Supporting these four values leads to some commonalities between the various Agile Methods. Instead of discussing the various differences and nuances of each methodology [13], this paper examines their commonalities from the perspective of the project manager. There are six common features to the various Agile Methods:[1] 1) collaboration, 2) code reviews, 3) small teams, 4) short release schedules, 5) time-boxing, and 6) constant testing.

All Agile Methods are highly collaborative, both inside of the development group and outside of it. Agile Methods rely on informal communication rather than voluminous documentation to rapidly spread information throughout the team and to other stakeholders. Without a highly collaborative environment, any Agile Method is doomed to fail. This means that a primary responsibility of the project manager is to ensure a highly collaborative environment. The project manager must be more of a coach and mentor than a dictator [13].

Agile Methods also encourage, if not require, code reviews. Code reviews allow for dissemination of key information. For example, in XP, code reviews are continuous through pair programming where two developers share a single computer [2].

Agile Methods also encourage small teams and small numbers of teams per projects. This ranges from a single team of three to sixteen developers on XP to up to six teams of two to six members on DSDM [13]. Small teams are required to foster collaboration, are more likely to require less process and planning to coordinate team members' activities.

Agile Method Release schedules can be as short as two weeks or as long as six months [13]— SCRUM even fixes the release schedule at thirty days [5]. At the end of each release, a functional product is released to the customer that allows for the product to be evaluated and for changes to be made in the priorities of features to be added to subsequent releases.

In time boxing, the release length is fixed but the features are not. This is the reverse of the "traditional" development mentality where the features are fixed and the delivery date flexible. Time boxing helps focus the customer and reduces gold-plating and scope creep.

To offset the potential for short releases to significantly degrade product quality, Agile Methods put a high degree of emphasis on testing the product throughout its lifecycle. The notion of test-first offsets the risks of a hacking mindset of just writing the code.

Agile Methods require integration testing throughout the development process. This testing must be automated with daily builds and regression tests to ensure all functionality works [13]. XP, DSDM, and SCRUM also include specific user acceptance testing at the end of a time-box, or even (with DSDM) concurrent with development [13].

[1] Not all agile methodologies have all six characteristics. SCRUM does not specify a development process and therefore does not explicitly require code reviews [9].

3. Impact on Project Management

While the ideas and intent behind Agile Methods are by and large good, they have impacts on the people, process, and project elements of an effort. We examine some of these impacts to determine whether an Agile Method can and should be applied to a project, given its requirements, available staff, and external factors such as business and legal constraints.

3.1 People

There can be a range of people involved in a software effort – developers, testers, project leaders to name a few. There is often a customer and an end user who wants the resulting product. There are also executive managers (business executives and directors of the development shop), who are interested in budgets and returns on investment, and human resources. Each of these has a stake in an agile project.

3.1.1 Developers

Perhaps the largest impact of Agile Methods is on the Developers. Agile Methods depend on strong developers – they must be amicable, talented, skilled, and able to communicate well [1]. Developers must be willing to work as a team, able to handle constant change, and resourceful enough to solve problems.

Agile Methods are very lightweight methods, not affording strict guidelines and processes for developers to follow. Hence, they do not accommodate weaker developers well. Yet, skilled technology workers are often a rare commodity. This is a management risk as some developers may not fit in this Agile environment.

Table 1. Boehm & Turner's developer levels [14]

Level	Characteristics
3	able to produce solutions in unprecedented situations
2	able to tailor solutions to fit new, but precedented situation
1A	solid developer able to implement functionality, estimate effort, & refactor code
1B	able to implement simple functionality, execute tests, & follow directions
-1	unwilling or unable to work in a collaborative environment

The "-1" level of developer depicted in Table 1, would be challenged in an agile environment. Even "1B" developers consume resources in "hand-holding" [15]. Hence, the top three levels make up the core of the agile development team. Boehm and Turner suggest level "3" developers may not be needed for all projects, depending on how unprecedented it might be.

Given the need for a high level of expertise, Agile Methods may be difficult to employ in a traditionally staffed organization. Highly skilled staff are always in demand, and without accommodating 1B developers, it may be difficult to build a long term human capital strategy. This is just one reason that long term projects present a significant risk for Agile Methods.

3.1.2 Testers

The impact of using an Agile Method on the testing (or quality assurance) organization hinges on the shorter development cycles where testing occurs throughout the development process [15]. Testers must work closely with the developers as code is being written. In Agile Methods such as XP, tests are changed before code is modified by the developers and the role of a tester is significantly reduced [2]. Testers focus on system and functional tests as more of an independent validation and verification role.

Testers may need to be more capable as programmers to automate their system and functional tests and incorporate them into the automated testing framework. This may represent a different skill set.

The project management challenge is to reallocate testers that no longer fit into the Agile group and find testers with appropriate development/testing skills. This represents an opportunity for novice developers (level "1B") to start, and gain system and Agile Method expertise. Such an approach requires one more experienced developer (or experienced test manager).

3.1.3 Project Leaders

There are two key Project Leader roles in software development – project managers and team leads. Each has a diverse set of challenges as management under an Agile Method differs from other methodologies. This distinction is well characterized as leading people and managing process resources.

Since Agile teams involve experienced staff with sizeable responsibility, a mentor or coach leadership approach is most effective. Team leads must be willing to enable members to take initiative. Leadership is done via collaboration rather than command and control type leadership [1]. This can represent a cultural shift for some as they must be willing to share decision making authority [8]. The job of a team lead is to facilitate the team into making decisions [2].

In contrast, project managers in agile processes are responsible for tracking progress and making business decisions. Project managers have a larger adjustment

than team leads since schedules and plans are far less important under agile methodologies. The emphasis is placed on responding to change rather than following a specific plan. This presents a challenge as they are usually called upon to detail the status of the project.

Project managers also have a much more involved role. In SCRUM, for example, the project manager meets with the team daily and leads the daily SCRUM [1]. Frequent/short meetings with the team are the norm for the Agile team [15]. Project managers are also more involved with the customer collaboration, instead of usual focusing on defining deliverables and contracts. If the project manager considering an Agile Method is not capable or does not want such a role, selecting an Agile Method may not be appropriate.

3.1.4 Customers

The impact of Agile Methods is to have customers much more involved than usual Methods. Customers in more traditional methodologies may be involved at the inception of the project – helping define requirements and contractual obligations – and at the end of the project with alpha, beta, and acceptance testing. Customers in Agile Methods are instead involved much more frequently and with more influence. Many Agile Methods assume, or at least highly recommend, a full-time customer presence on site working directly with the development organization [2][11]. Finding a customer willing to be this involved can be difficult. Commercial software companies may find customers unwilling to be involved. Startups may find customers are unknown as the market has yet to be defined. The availability of customer representatives must be considered when engaging the use of an Agile Method.

Merely having a customer representative available is not sufficient. They must be "committed, knowledgeable, collaborative, representative, and empowered" [11]. They must know what is required for end users. Also, since choices must be made about what features will be in what release, the representative must have the authority to make such decisions. Such a customer representative may not be available for all projects; hence Agile Methods may not be appropriate.

3.1.5 Executive Management

As with selecting any new organizational process, executive management support is essential [9]. For Agile Methods, this is particularly challenging as Executive managers are risk and opportunity focused – reluctant to induce risk without visibility. To justify expenditure, they want committed delivery dates for specific functionality, progress on tasking, and detailed schedules and plans. Agile Methods represent a major cultural change for them. With little documentation to track progress, features in a given release can change rapidly as the Agile process proceeds on-course.

Moreover, estimation of cost for a project specific function set is difficult under Agile Methods. Since requirements are not fixed, there is no way to know *a priori* what will be in a finished product and therefore when it might be finished. Executive management is faced with not being able to guarantee delivery dates, cost, or functionality [15] – a situation that is antithetical to most management approaches. This may be an untenable situation for management and prevent the adoption of an Agile Method in the organization.

The key for a project manager is to convince executives that Agile Methods will deliver faster and with better quality. If executive management is willing to give it a try, the success rate of projects using Agile Methods will determine its continued usage. For those project managers that build the trust with executive management to apply Agile Methods initially in appropriate projects, and win their confidence, the payoff for many projects can be high for the enterprise.

3.1.6 The Team

Since Agile Methods rely substantially on collaboration and communication, the team is key for success. A single strong-willed developer, developers who do not work well together, a customer who doesn't engage with the team, each could destroy the collaborative nature of a group. The team chemistry is of represents a significant risk for the Agile project.

Turnover is another significant personnel factor to be considered with an Agile team. Without formal documentation, high turnover on a project can lead to loss of critical knowledge. While this can be mitigated by code reviews and having developers rotate working on different functional areas, the loss of a significant member of a team can still be catastrophic. The project manager must consider this risk when examining whether the team (and the organization) is right for an Agile Method. Recognizing a key tenant for XP is retaining relevant knowledge by retaining good people.

3.2 Process

Since Agile Methods represent a new principles, processes activities, and sub-goals, they have an impact on many of an organization's processes. Old processes (e.g., planning, development, delivery, operations) must be replaced by agile ones. Cultural shifts in the organization towards Agile Methods turn old ways of thinking on their end, inducing resistance.

3.2.1 Planning

Agile processes are characterized by placing less emphasis on formal planning. This is not to say that planning does not occur. With so many small tasks, it is argued that agile processes require more planning, But unlike other methodologies, planning is not up-front followed by micro adjustments. Rather it is a constant task to ensure optimal delivery results [15].

Agile planning is a relatively informal process. For example, deciding what will go into each time-box is accomplished through the daily SCRUM meeting by discussing pending problems, prioritizing work, and assigning resources to the problems [12]. In other Agile methods, even this level of planning may not be considered [9]. It is important to factor in informality.

3.2.2 Documentation

In Agile Methods, documentation is sparse – often limited to source code and a set of user stories as in XP [2]. Most Agile Methods do allow for an optional architecture to be developed, and in DSDM it is even mandatory [13]. The driving factor for documentation is Agile Methods is how often it is going to change and need to be updated. A vision statement for a project might be extremely beneficial and never change – recording it warranted. Conversely, a low-level component design would be more likely to change, inducing some redundant document changes as well. This documentation-light process avoids wasted effort where documents are written once and then become obsolete as they are not updated to reflect the changes.

With Agile processes, information is communicated informally and is simply kept as part of the collective knowledge of the organization. While reducing the amount of documentation can increase productivity, it does come at some risk and cost. Documentation serves as a way to bring new members up to speed. It is useful when transitioning the project to a maintenance team. From a business perspective, documents form the basis for audits assuring proper quality procedures are followed. Documentation serves as a domain knowledge repository. If the organization changes dramatically, this knowledge can be lost.

3.2.3 Development Processes

Agile processes often encourage principles that dramatically change the process. While many of these are not limited to Agile Methods, Agile development encourages if not require their usage. Key development processes of interest are refactoring, minimalist development, code reviews, and continuous integration.

Refactoring is the process of taking code and improving it without loosing any functionality. Code might be improved for readability, maintainability, or performance. In refactoring, the code must pass all tests and abide by all defined contracts after it is rewritten. The development process question here is "when is refactoring prioritized over adding new functionality?"

Minimalist development within the Agile Methods community it is known as the YAGNI precept – an acronym for "You Aren't Going to Need It." Under YAGNI, features not needed for the current functional product are stripped out to keep the implementation simple. This reduces effort as well as "gold-plating" where unneeded functionality trickles in. The risk with YAGNI is that sometimes future requirements are known and building the system to support these requirements can lead to less effort down the road by eliminating major refactoring [11][16]. Projects that have well-defined future directions may not benefit from this aspect of the agile development process.

Code reviews are the process whereby one or more developers examines the code written by another. This could be continuous as in the pair programming aspect of XP [2] or be periodic as in the peer reviews incorporated in DSDM [13]. A key advantage of code reviews in this context is that they serve as a method of communication. Developers become familiar with the inner workings, design tradeoffs, and open issues with areas of the code they may be required to work with later. This can offset the risk of losing a member of the team, either temporarily on vacation or other leave or permanently due to a change in employment.

Continuous integration is the process whereby the system is tested often, usually nightly if not even more frequently [2]. Developers integrate their code into a baseline and run a set of regression tests on it. Continuous integration increases quality as side-effects of a change are quickly uncovered. Since finding defects early reduces the effort of fixing them, this aspect of the agile processes can have a significant impact on quality and schedules. However, developers must write a comprehensive set of tests to be used as regression tests and must take the time to integrate and test their code. This may require a shift in developer perspective if the developer is accustom to simply writing code which is then tested by a different group.

Many development groups already practice these principles. However, many developers are prickly about the notions of peer programming and may chafe at having to write a significant number of tests. It may be necessary for the project manager to incorporate these processes slowly and with incentives to increase the chances of their acceptance.

3.3 Project

Even the staunchest proponents of Agile Methods do not claim their universal applicability for projects. Different types of projects are more suited for Agile Methods than others. Business factors may prevent the utilization of Agile Methods. And there are several project characteristics that reduce the effectiveness and applicability of using Agile methods [10].

3.3.1 Project Types

Agile Methods are most applicable to projects where requirements are ill-defined and fluid since they seek to accommodate change easily. Projects that are unprecedented within an organization or use cutting edge technology (or are themselves the cutting edge technology) are examples of projects where change is likely to have a significant impact on the project.

Agile development does not lend itself to the types of rigorous analysis required to ensure the degree of assurance required for safety- and life-critical systems. This is because proving correctness is a non-Agile process requiring documentation and significant analysis. While quality is kept high by having a large number of tests, quality is only as comprehensive as the tests themselves. Code reviews are performed, but typically not with the rigor of formal methods that are more typically used with such critical systems.

3.3.2 Business Factors

A key business factor affecting the appropriateness of Agile Methods is contractual obligation. For many contracting companies, what is to be performed by the contractors is determined by a statement of work defining key requirements and tasking. If the requirements for work to be performed are part of a legal contract, an Agile Method may be inappropriate since requirements are malleable [2]. Documentation is also often used within a contracting relationship to indicate what work was done, document progress, and provide transition to the company contracting out the work. Government contracts often have significant documentation requirements such as adherence to standards like ISO9000. Changing this relationship to a less well-defined but more collaborative relationship may not be possible, at least in the short term.

Similarly, if the business requires the ability to specify release dates to accommodate customers, this represents a type of contract. Companies that use software products that are integral to their business, such as Enterprise Resource Planning (ERP) or financial applications, may need to know well in advance when a new version of a product will be released and what features will be contained in it so they can plan the migration to that version. Products that require a road-map for features that is well-defined may be unable to effectively use Agile Methods.

Documentation may also be required for regulatory reasons. For example, within the financial services community, the Securities and Exchange Commission requires certain types of documentation describing how issues such as insider trading and market timing are prevented. Such documentation might be able to be written and an Agile process still used, but the impact of change on this documentation must be considered.

3.3.3 Other Project Characteristics

Project time span is a significant characteristic that is an impact on the effectiveness of Agile Methods. Products that will take a long time to develop have risks that must be mitigated for an Agile Method to be used. It is likely that long-running projects will have a large amount of staff turnover over the duration of the project. Since Agile Methods rely on the collective knowledge of the team loosing members is a critical issue. This can be mitigated by rotating team members into different functional areas and by changing pair programmers periodically [14].

Long running projects are also a challenge since they tend to be larger in nature, with a high number of features and capabilities. This may lead to difficulties in prioritizing work. A single customer representative may not be sufficient and the project manager may be required to make decisions on priorities.

Another important aspect of long lived projects is that they tend to have long maintenance lives as well. Maintenance can be an issue for Agile development as the amount of documentation that can be used by the maintenance developers is often very small. It is also likely that the original developers have moved on and may not even remember the decisions that were made informally possible years before. Supporting a product that is expected to be in service for a number of years will probably require a degree of documentation significantly beyond that employed by Agile Methods.

The project roadmap is another key characteristic of a project to be considered. The YANGI principle can lead to significant rework that would not have been required had future requirements been considered initially [11]. An entire architecture may need to be rewritten to accommodate certain changes, especially to non-functional requirements such as performance and security [5]. Projects with roadmaps that are well-defined may benefit substantially from architectures that consider more than just the current release.

4. Conclusions

Agile Methods offer a reasonable approach for the high degree of change and uncertainty in today's software development. There are proven principles employed in Agile Methods that, when applied singularly under the right circumstances, result in lower risk projects and ultimately better productivity and quality (e.g., smaller teams result in lower risks due to the better communications). Additionally, when these are combined with other agile principles, there can be a synergy that provides even more traction on the project goals (e.g., small teams and pair programming result in fewer errors and less rework).

Since ungoverned software change can often be very costly, a methodology that addresses change can be a very useful tool for a project manager. However, Agile Methods are not appropriate for all projects. A project manager must consider the characteristics of the project to ensure that an Agile Method is appropriate. The impact on the people, the process, and the project must all be considered. For example, if a team of largely junior members is applied to a project that has very well understood requirements, and a mature software process is already in place in the organizations, there are three characteristics that argue against applying Agile Methods as a whole. However, the principle of small team might still be appropriate to reduce risks.

Furthermore, specific challenges with using an Agile Method can be offset by adding back some formality. For example, if migration to a separate maintenance group is required, documentation could be written by the development group as part of the transition [5][10].

Agile Methods offer software project managers an alternative development and management methodology that provides good support for projects with ill-defined or rapidly changing requirements. Even on project that are questionable for the application of the entire Agile Method, underlying agile principles may still be effective. Project managers should consider its usage for such projects assuming that they have a team capable of using it and can implement the required processes. Otherwise, more traditional approaches may be more appropriate.

5. References

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

[2] K. Beck. *eXtreme Programming Explained*. Addison-Wesley, 2000.

[3] "SCRUM, it's about Common Sense., http://www.controlchaos.com/about

[4] J. Stapleton. *DSDM – Dynamic System Development Method*. Addison-Wesley, 1995.

[5] Paetsch, F., A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development," *Proceedings of the 12th IEEE international Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, June 2003, pp. 308 – 313.

[6] D. Phillips. *The Software Project Manager's Handbook: Principles that work at Work,* IEEE Computer Society Press; June 1998.

[7] "Manifesto for Agile Software Development," http://www.agilemanifesto.org

[8] Williams, L. and A. Cockburn, "Agile Software Development: It's about Feedback and Change," *Computer*, June 2003, pp. 39-43.

[9] Cockburn, A. and J. Highsmith, "Agile Software Development: The People Factor," *Computer*, November 2001, pp. 131-133.

[10] Abrahamsson, P., J. Warsta, M.T. Siponen, and J. Ronkainen, "New Directions on Agile Methods: A Comparative Analysis," *Proceedings of the 25th International Conference on Software Engineering*, May 2003, pp. 244-254.

[11] Boehm, B., "Get Ready for Agile Methods, with Care," *Computer*, January 2002, pp. 64-69.

[12] Derbier, G., "Agile Development in the Old Economy," *Proceedings of the Agile Development Conference*, June 2003, pp. 125-131.

[13] Thomas, S. "An Agile Comparison," http://www.balagan.org.uk/work/agile_comparison.htm.

[14] Boehm, B. and R. Turner, "Using Risk to Balance Agile and Plan-Driven Methods," *Computer*, June 2003, pp. 57-66.

[15] Cohn, M. and D. Ford, "Introducing an Agile Process to an Organization," *Computer*, June 2003, pp. 74-78.

[16] DeMarco, T. and B. Boehm, "The Agile Methods Fray," *Computer* , June 2002, pp. 90-92.