An Agile Software Development Framework

Malik F. Saleh
msaleh@pmu.edu.sa
Management Information Systems
Prince Mohammad Bin Fahd University
Al Khobar, 31952, Saudi Arabia

Abstract

Agility in software projects can be attained when software development methodologies attain to external factors and provide a framework internally for keeping software development projects focused. Developer practices are the most important factor that has to cope with the challenges. Agile development assumes a project context where the customer is actively collaborating with the development team. The greatest problem agile teams face is too little involvement from the customer. For a project to be agile, the developers have to cope with this lack of collaboration. Embracing changing requirements is not enough to make agile methods cope with business and technology changes. This paper provides a conceptual framework for tailoring agile methodologies to face different challenges. The framework is comprised of three factors, namely, developer practices, customer collaboration, and predicting change.

Keywords: agile framework, tailored framework, software development, developer practices, customer collaboration, predicting change

1. INTRODUCTION

Different software development approaches share common principles such as improving customer satisfaction, adopting to changing requirements, delivering working software, and creating a collaboration among stakeholders and developers [1]. An investigative study by [2] found evidence of staffing stability and design compatibility that affect the success of software projects that use incremental and iterative development. The basic idea behind iterative development is to develop a software system incrementally, allowing the developer to take advantage of what was learnt during the development of earlier deliverable versions of the system. Therefore, iterative and incremental software developments are viewed as the cornerstone of the agile methodologies [3].

One of the main reasons for using agile methodologies is to satisfy the needs of the users. The informal communication among stakeholders and developers sometimes raises problems such as inability to cope with system complexity and rapidly changing requirements, or inability to add new members to the development team [4]. Agility can be attained when software development methodologies attain to external factors and provide a framework internally for keeping software development projects focused. Those methodologies can be applied in different units of the organization with minor modifications. It is important to note that there is no best-fit methodology for an organization [5].

Agile methodologies appeared as a reaction to traditional ways of developing software and acknowledged the need for an alternative to documentation driven, heavyweight software development processes. Different agile methodologies are in use such as Extreme Programming, Scrum, Cockburn's Crystal family, and Feature Driven

Development. These different methodologies have less documentation and more code-oriented features that are common among them [6]. Extreme Programming (XP) tends to be best accepted by the developers [7].

Extreme Programming is the most popular of the various types of agile methodologies. It takes many of the best well-known software development practices and applies them during system development. XP is a set of values, principles and practices for rapidly developing high-quality software by preaching the values of community, simplicity, feedback and courage [6, 8]. According to [8] XP has twelve core practices (see Table 1).

Table 1: Extreme Programming Core Practices

	Description
Planning Game	The customer and development teams decide on what features will be of implemented
Simple Design	The simplest design that meets the business requirements is selected
Small Releases	Release a useful set of features into production and update them frequently in short cycles
Metaphor	Choose system names that helps you understand the parts of the system and help you communicate more effectively
40-Hour Week	Maintain productivity and avoid burn out of the development team
On-site customer	On-site customer ensures effective communication as a result less documentation will be required
Coding standard	Coding standard is a way of communication and documentation
Continuous Integration	frequent code integration means less chances that there will be diversion and it will result in more testing
Refactoring	Applying small updates in small steps reduces the risk of introducing errors
Pair Programming	Programmers are paired to write code using a single machine. This helps the code to be constantly reviewed while being written and will produce high quality code with little decrease in productivity
Collective Code Ownership	The code belongs to every member of the team
Testing	Test-driven development. Code is validated at all times and before new features are added

In this paper we provide a conceptual framework for modifying the agile methodologies to cope with changing requirements. This model is tailored to the needs of the developers and the needs of the software project which makes the agile methodologies themselves flexible. In section 2 we discuss the related work that enhances or negatively affects agile methodologies. Section 3 introduces the motivation for this paper and the framework for improving agility in software development. Section 4 discusses the conclusion and our future work.

1. BACKGROUND AND RELATED WORK

Little research has been undertaken into what is meant by agility and how a supposed agile method can be evaluated with regard to its agile approach [9]. Agile methods are labeled as agile because of their ability to handle changing requirements. It is also expected that the agile methods themselves are flexible and can be tailored to the needs of the developers and the needs of the software project. There is also acknowledgement in the literature that software methods should be tailored if they are to achieve optimum effect [10]. Researchers believe that

flexibility is one of the key selling points of agile methods [8]. The suitability of more or less agility or planning depends on the context of development as well as on the kind and size of the software to be developed [11]. Adoption of agile methods seems to need an all-or-nothing approach. They are often welcomed by both managers and programmers as providing a much needed release from the overheads typically perceived as being imposed by traditional software development approaches [12]. But managers are faced with challenges. Software development managers may be unsure how to adopt agile methods incrementally and which approach to choose as most appropriate for their situation [13]. Another challenge for the managers is how to ensure that their adopted method can mature and grow as the development team's skills mature and how to ensure that the development team doesn't resist change [12].

Various agile development methods include documented procedures that involve tasks and milestone, and product development strategies that involve requirements, designs, and architectural plans. Compared to unplanned methods, agile methods emphasize a fair amount of planning. Their general view places more value on the planning process than the resulting documentation, so these methods often appear less plan oriented than they really are [11]. While the agile methods have less documentation and more code-oriented feature they are knowledge-intensive. Knowledge is gained and shared within the different aspects of the methodology.

There are four factors that determine the success of software development projects: Quality is to deliver a good software product to the customer or a project outcome as perceived by all stakeholders. Scope is meeting all requirements and objectives that the customers wants in the software. Time is delivering the final product to the customer on time. Delays in delivering releases, as long as the final product is delivered on time, will not affect this factor. Finally, the last factor is cost which is delivering the product within the estimated cost and effort [14].

A framework was developed by Qumer and Henderson-Sellers which has four areas to crystallize the key attributes of agility. The framework has the following dimensions: flexibility, speed, leanness, learning and responsiveness. Flexibility is the ability to respond to change and leanness accentuates lower cost, reduced timeframe and quality production [9].

Another framework was developed by [10] that is comprised of two factors that can improve method tailoring effectiveness. The first area is the characteristics of the method, and the second is developer practices. The framework was developed to overcome some of the problems traditionally associated with software development's tendency to replace older methods with new apparently improved alternatives. In addition, the framework assesses how amenable Extreme Programming is to tailoring, and it developed a set of recommendations for its improvement.

1. MOTIVATION AND THE CONCEPTUAL FRAMEWORK

A decade ago, a group of software practitioners agreed on some software development principles. The Manifesto was launched and it was a breakthrough that had implicit goals of producing working software that values meeting the requirements of the customer while having little documentation. Therefore, the Manifesto inspired developers to produce software that was responsive to customers' needs and employ a light-weight development methodology.

In this decade, there is a shift from producing working software to improving the experience of the customer with the software and instead of responding to change, agile developments have to predict the change. The third shift in software development is that the software is developed before the customer asks for it. Companies develop software and they compete for the customer. Therefore, collaboration with customers is not possible. In effect, companies have to find the customer after the software has been developed.

The motivation for this research is that agile methodologies are flexible enough to be adopted by software developers but they need tailoring to provide a one size fit-all approach. We proposed a conceptual framework drawn from the existing method tailoring literature. The main pillar of agile methodologies involves the interaction between developers and customers. The greatest problem agile teams face is too little involvement

from the customer. For a project to be agile, the developers have to cope with the challenges. The framework is comprised of three factors that can improve agile method tailoring effectiveness and they are used to cope with challenges in agile methodologies, namely, developer practices, customer collaboration, and predicting change.

This conceptual framework contributes in three areas: first, it provides a tailored approach to agile methods. Second, it provides a solution for a one size fit-all approach for agile methods. Finally, it provides flexibility in choosing agile methods without following the exact principles that agile methodologies require from the development team.

3.1 Developer Practices

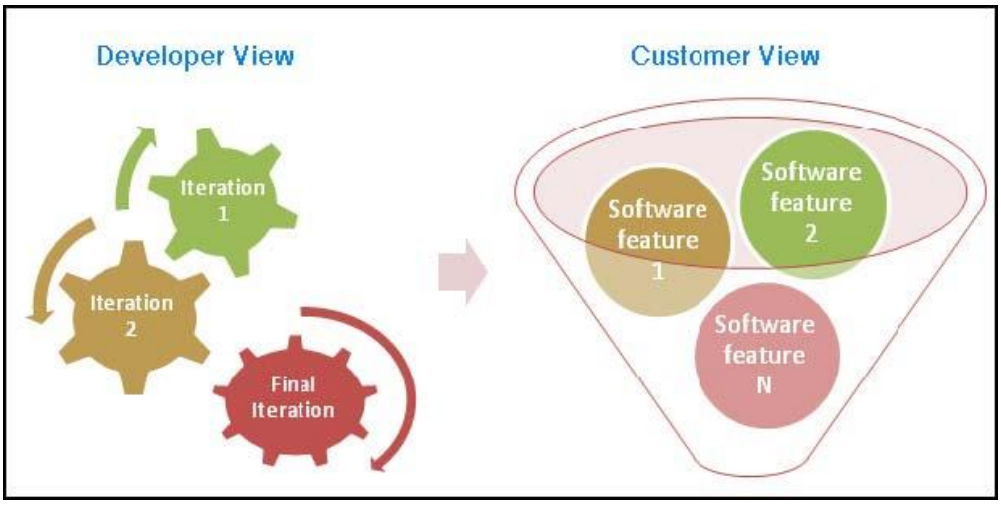
Agile methods are often seen as providing ways to avoid overheads typically perceived as being imposed by traditional software development environments [12]. The fact that agile methods are focused on the people rather than on reporting deliverables is often seen as a welcome shift of balance towards the most important factor in software development: the personnel involved [9]. Agile methods actively involve the customer in the development process. However, it is important to consider the involvement and input from all stakeholders such as partners, and suppliers. In addition to external sources, other business units within the organization should be involved [15]. Two aspects of the personnel involved in the development of agile software projects are highlighted in this framework: The developer and the customer.

Developers are likely to face challenges as they are required to change their work habits and acquire new skills. The developers should work in self-organized teams. Where each leader and his team members decide which task to work on or how a problem will be solved. This self- organizing structure will encourage the developers to fully own the problem and provide the best solution for it. The team should be cross-functional so that everyone can take a feature from the project idea and have its implementation. It is important to remember when selecting an appropriate team structure that it is not permanent. While you don't want to continually change team structures, if the current structure is clearly wrong, change it [16].

The team must select an iterative and adaptive approach to complete the project. The team will decide the length of the iteration and the list of potential goals for that iteration. Each member of the team is asked for the list of needed resources and an estimate to complete the tasks. The list of goals comes for the customers, the team members, and the project leader. Team members need to brainstorm to find the details of the tasks and how to implement them.

The iterative approach depends on an adaptive planning. Iterations in the development are to be adapted based on feedback from all stakeholders rather than predicting the project plan. As a result, each development's iteration will create a plan for the next iteration that will adjust as the project features unfold. This approach does not eliminate project objectives and milestones, rather, it encourages flexibility in planning the entire projects details. In addition, the team will manage the details of the projects while the customer is involved in the project features that will be delivered. Figure 1 illustrates this approach.

Figure 1: Developers iterations and customers view



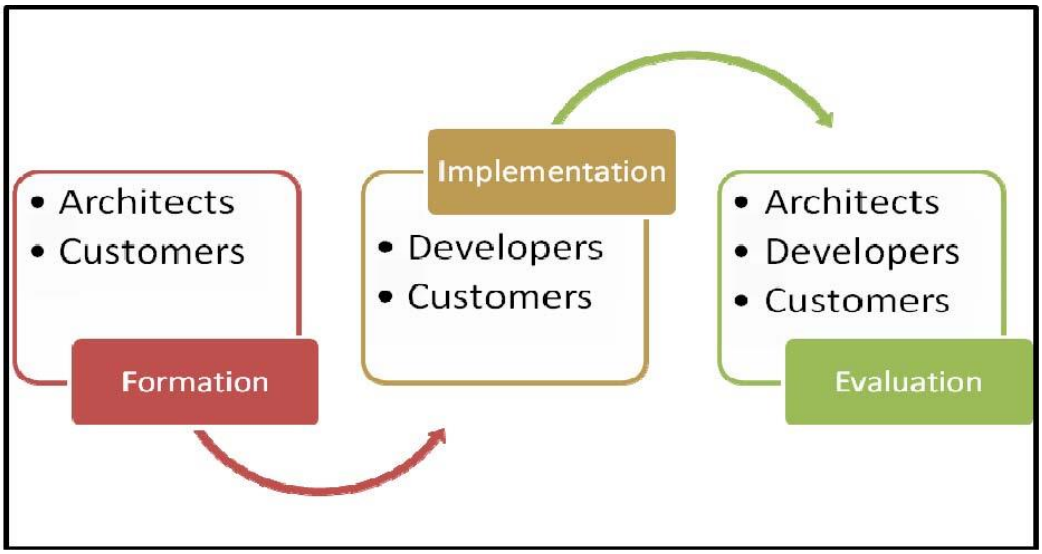
Each leader of a team is a member of the other teams that affect the group that he/she is leading. So, output from one team is delivered as input to another team and the team leader is ensuring that the implementation is according to the plan.

The development team should separate the interfaces from implementations. The responsibilities between developers and architects should be divided around the interfaces and implementations. Collaboration should occur between these two groups. Architects should become more like developers with respect to accepting changes that occur within the life cycle of a system. Developers should be given standard interfaces to work with that were created by interface design experts and finally developers shouldn't be overwhelmed with too much architectural complexity [17]. Therefore, the first practice that the agile team should adopt is dividing responsibility to allow the developers to focus on Implementations.

The team structure depends on the organizations and the project. Large teams may include members with more diverse skills, experiences, and approaches and, as such they are not as much at risk to the loss of a key person. In addition, large teams provide more opportunities for individuals to specialize in a technology or a subset of the application. On the other hand, there are even more advantages to a small team that would support agile methods. In small teams, developers need less time coordinating the efforts of the team members and an individual developer is less likely to take on distinct roles [16].

The development of software by large teams of developers requires a steady flow of elicited requirements. Without this steady flow of requirements, the project run the risk of delaying new software iterations and bad code due to badly specified requirements, all resulting in the waste of large amounts of resources. For the development to remain agile, the requirements are defined and implemented quickly, while others move through their lifecycle at a regular pace [18]. Whether the project consists of large teams or small teams the responsibilities are divided and three processes are created: Formation of the system, implementation by the developers and evaluation by the developers, the architects, and the customers (See Fig. 2).

Figure 2: Dividing and sharing responsibilities



The interaction between the customer and the development team is a vital feature and an important success factor in agile software development [19-21]. Agile methods expand the customer role within the entire development process by involving them in writing user stories, discussing product features, prioritizing the feature lists, and providing rapid feedback to the development team on a regular basis [21].

A study found that the greatest problem agile teams face is too little involvement from the product owner or customer [20]. For a project to be agile, the developers have to cope with the challenges. The following agile approach is used to cope with challenges:

- Communication Challenges: e-Collaboration is to be used when the customer is having little involvement in the project or the involvement is not on a regular basis.
- Clarification Challenges: When developers are waiting for scenarios, stories, or clarification from the customer, the development stops until the developers receive the required customer's clarification. Therefore, the solution is changing priority to keep the project on time and within the scope. They have to change the priorities of the features and the iteration.
- Onsite Customer Challenges: A situation is faced where the developers don't have an on-site customer. One of the developers is chosen to play the role of a fulltime customer or an agile Tester. His role is to develop and run system tests and develop and run acceptance tests. This developer will coordinate with the customer to provide stories to the development team.

Agile methods actively involve the customer in the development process. For agile projects, the customer is not only the one requesting the software, it also includes partners, suppliers, and other business units inside the organization. The developers in the project internally need to gain external knowledge and ideas about the project that they will develop.

3.2 Collaboration

The involvement of team members in a project depends on the information provided. Communication plays a fundamental factor in the success of a project. Communication according to [22] acts as the glue that links together all work by the team members. Everyone should communicate with everyone else. Problems arise if the team leader withholds information from the developers. The project leader should provide mentoring and sharing of relevant project information with all members. While the leadership role is important to coordinate and lead the project, it is more important to create a culture that fosters a collaboration, rather than communication between all team members. Additionally, developers are more involved in the project if the team members share relevant

project information with each others, not just the project leader. Therefore, how the work is coordinated in the project is an important factor that affects the success of the project.

Another success factor that affects the project is the feedback that team members receive from each other. Developers can benefit from this feedback to improve their skills and to produce better code that has been shared with others. Therefore, when pair programming is employed each person in the team shares the ownership of the code and the feedback improves the skills of the other team member. When the entire team participates in giving feedback to other members, a group ownership of the code will be employed and as a result the skills of every team member are improved. If the skills of all the developers are enhanced, developers can collaborate by face-to-face communication, which is the best method of conveying information, or they can collaborate in the code that they write. Code communication is essential for group ownership of the project. Developers have to understand the code that was written by others in order to modify it if there is a need. Another essential enhancement for the collaboration among team members is to provide cross training. Cross training enhances the development and creates a shared team-interaction model [23].

The greatest problem agile teams face is too little involvement from the customer. What if the customer does not exist when the software was being developed? It is a normal business practice for software development companies to develop software first and then compete for the customer. Companies find the customer after the software has been developed therefore collaboration with customers is not possible.

For agile methods to be used with software projects that don't have a specific customer, tailoring is needed. First, requirement gathering has to employ survey and brainstorming techniques to gather the requirements. Survey is conducted based on potential customers and brainstorming is conducted among the team members. One of the developers, or a group, is chosen to take on the distinct role of the customer. The role of this developer will have the story telling role of the customer, running system tests and running acceptance tests.

Finally, the agile Manifesto values individuals and interactions over processes and tools, working software over comprehensive documentation, and customer collaboration over contract negotiation [24]. While collaboration is an important principle, e-collaboration can be used when the customer cannot be onsite. Therefore, agile projects have to create a culture where everyone is putting ideas in the project as well as taking them out regardless of the medium being use. Collaboration can be face-to-face, phone conferencing, web conferencing, emails, or using the many e-collaboration tools.

3.3 Predicting Change

When implementing change, different method for requirement elicitation can be used. Interviews are widely used when the developers elicit information from system experts and each other about the structure and the behavior of the system [25]. Agile methods embrace changing requirements, even late in the development cycle. Projects most responsive to change will offer support for future changing requirements. Other than requirement changes, software face two types of change: technology infrastructure and business processes. Utilizing of technology requires alignment of technology strategies with the business strategies. This alignment reflects the view that the business success depends on the linkage of business strategy with information technology.

When faced with the challenge of predicting future changes in technology, an agile team has to distinguish between operational technology and strategic technology. The acceptance of new technologies by their intended users persists as an important issue for the developers. Therefore, support for new technology should be built-in in the design of the software. For example developers should use software components that support the latest technology. Developers use reusable components to produce high quality software systems. These systems need to satisfy not only the initial demands of the customer, but they need to also offer support for future, changing requirements [26]. A good example to support this idea is providing applications that run on different operating systems in the Smartphone ecosystems.

Finally, software development should focus on the business models and business processes before they focus on technology infrastructure or applications. The software team has to be aware of market trends when implementing software projects. The software industry shifted from the client-server to the distributed model then from the services oriented architecture to the Cloud. Therefore, keeping an eye on demand for future trends will make the transition to new changing requirements manageable for software developers.

1. ANALYSIS OF THE TAILORED FRAMEWORK

One of the main reasons for using agile methodologies is to satisfy the needs of the users and to address the issue of changing requirements. Agile developments introduced new concepts for effective communication such as on-site customer to ensure effective communication with the developers. As a result of this communication, less documentation is required for agile projects.

Ultimately, communication is at the center of solving problems in agile developments. However, certain agile development projects cannot have an on-site customer because the project is being developed without a customer or for other reasons. Tailoring of agile developments to satisfy the needs of different situations is therefore required to keep the communication at the core of practices for agile methods. While the argument is that agile development cannot be used without a customer, tailoring of the agile methodologies is proof that development projects can use agile methodologies even if the project does not have an on-site customer. Other communication challenges can be solved in different ways. E-Collaboration is to be used when the customer is having little involvement in the project or the involvement is not on a regular basis. While the developers wait for scenarios, stories, or clarification from the customer, the development stops until the developers receive the required customer's clarification. The solution is to change priorities to keep the project on time and within the scope

Following a coding standard is another way of communication among developers and for documenting the project. Communication through a coding standard is not affected because this is an internal factor and it is not affected by the different situations that the projects might face. Therefore, a coding standard is enforced and developers have to use a coding standard that can be understood by all members of the development team.

Some projects require small teams while others require large teams with diverse skills and experiences. Large teams provide opportunities for a developer to specialize in a technology or a subset of the application. One of the core practices of agile methodologies is collective code ownership. Developers who specialize in a technology or a subset of the application must share their knowledge with the development team to facilitate the collective code ownership practice.

Projects most responsive to change will offer support for future changing requirements. Software projects face two types of change: technological infrastructure and business processes. Utilizing of technology requires alignment of technology strategies with the business strategies. Agile projects are mandated to keep the developers aware of the latest technology changes and developers are mandated to implement the latest technology.

Therefore, agile methodologies are flexible to cope with external factors such as changing requirements while the core practices are not affected. Coping with external factors requires flexibility in the developer practices without changing the core practices of agile developments. Core practices that are internal in the projects are not affected by the different situations that agile projects face.

1. CONCLUSIONS

Projects don't have to follow a formal, well-established methodology to be successful in software development projects. Most agile methodologies institutionalize the same set of practices. It often does not matter which one you choose, as long as you follow it diligently. Areas that agile development focuses on are the communication among all stakeholders, iterations with continuous integration, and feedback. One of the main reasons for using agile methodologies is to satisfy the needs of the users. The informal communication among stakeholders and

developers sometimes raises problems such as inability to cope with system complexity and rapidly changing requirements. Agile methods are labeled as agile because of their ability to handle changing requirements. It is also expected that the agile methods themselves are flexible and can be tailored to the needs of the developers and the needs of the software project.

Agility is attained when software development methodologies attain to external factors and by being flexible internally in software development. This framework proposed three factors for making agile frameworks agile in their practices. We showed evidence that developers practices and eliciting the requirements play a major role in making the software development successful. Therefore this conceptual framework created two customized practices for making agile methodologies handle different situations when the original practices do not accounting for these situations.

The third factor was introduced because there is a shift from producing working software to improving the experience of the customer with the software and instead of responding to change, agile developments have to predict the change. The third shift in software development is that the software is developed before the customer asks for it. Companies develop software and they compete for the customer. Therefore, collaboration with customers is not always possible.

1. REFERENCES

1. Paetsch, F., A. Eberlein, and F. Maurer, *Requirements Engineering and Agile Software Development*, in *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*. 2003, IEEE Computer Society. p. 308.
2. Tan, T., et al., *Productivity trends in incremental and iterative software development*, in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*. 2009, IEEE Computer Society. p. 1-10.
3. Larman, C. and V.R. Basili, *Iterative and Incremental Development: A Brief History.* Computer, 2003. **36**(6): p. 47-56.
4. Uikey, N., U. Suman, and A.K. Ramani, *A Documented Approach in Agile Software Development.* International Journal of Software Engineering (IJSE), 2011. **2**(2): p. 13-22.
5. Kar, N.J. *Adopting Agile Methodologies of Software Development*. 2006; Available from: http://www.infosys.com/infosys-labs/publications/Documents/adopting-agile-methodologies.pdf.
6. Agarwal, R. and D. Umphress, *Extreme programming for a single person team*, in *Proceedings of the 46th Annual Southeast Regional Conference on XX*. 2008, ACM: Auburn, Alabama. p. 82-87.
7. Ilieva, S., P. Ivanov, and E. Stefanova. *Analyses of an agile methodology implementation*. in *30th EUROMICRO Conference* 2004. Rennes, France: IEEE Computer Society.
8. Beck, K., *Extreme Programming Explained: Embrace Change*. 2000, Reading, MA: Addison-Wesley.
9. Qumer, A. and B. Henderson-Sellers, *An evaluation of the degree of agility in six agile methods and its applicability for method engineering* Information and Software Technology, 2011. **53**(5): p. 509-520.
10. Conboy, K. and B. Fitzgerald, *Method and developer characteristics for effective agile method tailoring: A study of XP expert opinion.* ACM Trans. Softw. Eng. Methodol., 2010. **20**(1): p. 1-30.
11. Boehm, B., *Get ready for agile methods with care.* Computer, 2002. **35**: p. 64-69.
12. Qumer, A. and B. Henderson-Sellers, *A framework to support the evaluation, adoption and improvement of agile methods in practice* Journal of Systems and Software, 2008. **81**(11): p. 1899-1919.
13. Syed-Abdullah, S., M. Holcombe, and M. Gheorge, *The impact of an agile methodology on the well being of development teams.* Empirical Software Engineering 2007. **11**: p. 145–169.
14. Chow, T. and D.-B. Cao, *A survey study of critical success factors in agile software projects* Journal of Systems and Software, 2008. **81**(6): p. 961-971.
15. Conboy, K. and L. Morgan, *Beyond the customer: Opening the agile systems development process* Information and Software Technology, 2011. **53**(5): p. 535-542.
16. Cohn, M., *Succeeding with Agile: Software Development Using Scrum*. 2009, Redwood City, CA: Addison-Wesley Professional.
17. Gall, N. and A. Bradley, *Best Practices for Dividing Developer and Architect Responsibilities to Achieve SOA-Based Agility.* Gartner, 2099.
18. Vlaanderen, K., et al., *The agile requirements refinery: Applying SCRUM principles to software product management* Information and Software Technology, 2011. **53**(1): p. 58-70.
19. Chow, T. and D. Cao, *A survey study of critical success factors in agile software projects.* Journal of System Software, 2008. **81**: p. 961–971.
20. Hoda, R., J. Noble, and S. Marshall, *Agile undercover: when customer's don't collaborate*. 2010, Springer: Norway. p. 73–87.

21. Hoda, R., J. Noble, and S. Marshall, *The impact of inadequate customer collaboration on self-organizing Agile teams* Information and Software Technology, 2011. **53**(5): p. 521-534.
22. Moe, N.B., T. Dingsøyr, and T. Dybå, *A teamwork model for understanding an agile team: A case study of a Scrum project.* Information and Software Technology, 2010. **52**(5): p. 480-491.
23. Marks, M.A., et al., *The impact of cross-training on team effectiveness.* Journal of Applied Psychology, 2002. **87**(1): p. 3-13.
24. Manifesto. *Principles behind the Agile Manifesto*. 2001 [cited 2011 August]; Available from: http://agilemanifesto.org/.
25. Lindvall, M. and K. Sandahl, *How Well do Experienced Software Developers Predict Software Change?* Journal of Systems and Software, 1998. **43**(1): p. 19-27.
26. Vaucher, S. and H. Sahraoui, *Do software libraries evolve differently than applications?: an empirical investigation*, in *Proceedings of the 2007 Symposium on Library-Centric Software Design*. 2007, ACM: Montreal, Canada. p. 88-96.